Alignements locaux pour la reconnaissance de repliements des protéines par programmation linéaire en nombres entiers

Guillaume Collet

▶ **To cite this version:**

Guillaume Collet. Alignements locaux pour la reconnaissance de repliements des protéines par programmation linéaire en nombres entiers. Modélisation et simulation. Université Rennes 1, 2010. Français. NNT : . tel-00512725

HAL Id: tel-00512725
https://theses.hal.science/tel-00512725

Submitted on 31 Aug 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

N° d'ordre : 4166 **ANNÉE 2010**

THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Informatique

Ecole doctorale MATISSE

présentée par

Guillaume Collet

préparée à l'unité de recherche IRISA
Institut de Recherche en Informatique et Systèmes Aléatoires
IFSIC

Alignements locaux pour la reconnaissance de repliements des protéines par programmation linéaire en nombres entiers

Thèse soutenue à Rennes le 8 juillet 2010

devant le jury composé de :

Dominique Lavenier
Professeur - ENS Cachan / *président*

Jin-Kao Hao
Professeur - LERIA / *rapporteur*

Alexandre G. De Brevern
Chercheur - INSERM / *rapporteur*

Michel Masella
Chercheur - CEA / *examinateur*

Rumen Andonov
Professeur - IRISA / *directeur de thèse*

Jean-François Gibrat
Directeur de recherche - INRA / *directeur de thèse*

« La vraie réussite, c'est aller d'échecs en échecs sans perdre son optimisme. »
Gustave Parking

Remerciements

Les étapes de ma vie s'enchaînent les unes après les autres et je ne m'en soucie pas, mais pour celle-ci, je vais faire une pause et remercier quelques personnes.

En premier lieu, je veux remercier mes parents car ils m'ont toujours incité à comprendre les choses et ne m'ont jamais empêché d'avoir des explications... même sur le nombre Pi. Je les remercie de m'avoir laissé choisir ma voie, au risque de me perdre, ce que j'ai fait quelques fois, mais ça en valait la peine.

Je tiens à remercier Cathy, il y a beaucoup d'elle dans cette thèse. Partager ma vie n'est pas toujours facile, surtout quand je déprime et que je veux tout arrêter. Cathy a supporté mes grognements et mes rires, mes nuits blanches et mes ronflements, mes déceptions et mes succès. Pour avoir supporté tout cela et surtout pour avoir toujours cru en moi, je te remercie du fond du cœur.

Je voudrais maintenant remercier les gens que j'ai rencontrés au cours de mes études et qui ont, involontairement peut-être, participé à la réalisation de cette thèse. Ils sont aujourd'hui des amis même si je ne les vois plus aussi souvent qu'avant. Par ordre de rencontre (à peu près) : Rahan, Kaelig, Momo, Seb, Psyko, Tahau, Nadège, Céline, Alex, Rozenn, PF, Tom... et j'en oublie sûrement. Merci à tous pour les bons moments, les soirées étudiantes, les discussions enflammées, les jeux de mots débiles, la caïpirinha, les après-midi crèpes, les jeux de société, dance dance révolution et surtout votre amitié.

Après, il y a le boulot... Merci à l'équipe Mig et à l'INRA de Jouy en Josas pour leur accueil, leur machine à café de compétition, les croissants et les pains au chocolat, la cantine (la crême brulée me manque), et les Tlags qui m'ont propulsé sur le devant de la scène pour la première fois. J'en profite pour remercier chaleureusement Antoine Marin, qui m'a encadré lors de mon stage de master, et Jean-François Gibrat, qui m'a encadré au cours de ma thèse.

Je remercie également l'équipe Symbiose, chez qui j'ai passé de très bons moments au cours de ces quatre dernières années. Merci à vous pour vos conseils, votre aide pour ma soutenance, les discussions, sérieuses ou non, les pauses cafés lancées de main de maître par Noël, les jeux de mots de François, les pots, les barbecues, les séminaires "Aubade", les SAV. Je remercie en particulier les collègues qui ont partagé le bureau

avec moi, ça ne devait pas être facile tous les jours. Merci donc à Christine, Olivier et Nolwenn. Je remercie enfin Rumen Andonov pour m'avoir permis de réaliser ce travail.

À l'IRISA, il n'y a pas que l'équipe Symbiose, et je me dois de remercier chaleureusement Pierrette, Laurence et les ptits jeunes de la cafet. Un café servi avec le sourire, c'est rien du tout, mais c'est tellement plus que ça ! Je remercie aussi tous les joueurs de percus de la batucada et les joueurs de go... les pauses, ça fait du bien. Je remercie du même coup Marie qui nous a fourni le matériel pour jouer et qui est d'une gentillesse incroyable. Enfin, je remercie les amis que j'ai croisés en ces murs, dans le désordre : Xavier, Olivier, Sylvain, Matthias, Guillaume, Alex, Sophie, Noël, Julien, Goulven, Thibaut... Un merci spécial à Xavier qui m'a fourni le modèle LaTex pour rédiger ma thèse et qui m'a également appris à jouer de l'accordéon diatonique même si je n'étais pas très appliqué.

Il y a une personne que je ne peux pas oublier dans ces remerciements (et si il les lit, il doit commencer à se poser des questions), c'est Guillaume Launay. Je crois, sans exagération, que je n'aurais pas fini ma thèse sans ton aide. Te remercier est donc la moindre des choses : Merci beaucoup *Old Chap'*.

Je veux aussi remercier les copains avec qui j'ai joué de la musique. Les HeadStuff/Dicotylédones : Armelle, Jacques, Olivier, Cécile, Valérie et Hugues. Les membres de Bloco Loco : Raphaël, Blandine, Loïc, Arnaud, Mireille, Olivier, Sabine, Brice...

Enfin, comme tout bon sportif sous contrat, je vais maintenant citer mes sponsors que je remercie : ma famille, c'est-à-dire ma sœur, mon beau-frère, ma belle-mère (fournisseuse officielle d'eau minérale), mes tantes, oncles, cousins, cousines, bref tous ceux qui m'ont soutenu de loin comme de près, les gars du helpdesk Irisa, l'école de danse El Divino, Apple, la dame qui fait les crêpes au Géant-Casino, l'ANR Proteus, les bonbons Haribo (Dragolo exclusivement, j'aime pas les réglisses), Omar et Fred, Mozinor, les Robins des Bois, la boulangerie de M et Mme Geslin, TEM la firme, Puzzle Bubble, l'inventeur des origamis, le Fuji et le Sushi Shop, Terre de Jeux, les librairies Critic et M'enfin, les Shadoks, le Blog du Mac, Nutella, Happy Tree Friends, les TED Talks, Gary Burkhart et Garr Reynolds.

Et enfin, enfin, je remercie ceux qui me soutiennent au quotidien, toute ma compagnie créole : Séraphine, Mercredi, Patton, Diabolo, les Frites, les Elvis et les Francis, Gris-Sel, Joel et Picoti, Petit Joel, Courgette, le Piou, Shogun et le ptit singe, Pirate, les Cochons, Cocotte et son gros œil, Les ptites Personnes, et surtout mon ptit cœur.

Merci à tous.

Table des matières

Introduction

L'Annotation des génomes

La génomique est une science en plein essor qui soulève de nombreuses questions de recherche et s'accompagne de progrès technologiques fulgurants. Le décryptage des génomes n'est pas seulement un sujet d'actualité mais également un vaste champ d'expérimentations, de questions non résolues, bref de recherche.

Grâce aux progrès technologiques de ces dernières années, environ 200 nouveaux génomes complets sont séquencés chaque année. Fin 2009, nous comptons environ 1100 génomes complets référencés dans GenBank [8] et le nombre de génomes séquencés devrait continuer à croître fortement grâce aux séquenceurs de nouvelle génération (NGS). La compréhension du fonctionnement des organismes passe, en partie, par l'analyse de leur génome. Cette analyse consiste à déterminer un certain nombre d'informations comme la position des gènes, les protéines produites par les gènes, les fonctions de ces protéines, et enfin leurs interactions. Cette analyse est appelée *annotation* dans le sens où il s'agit d'annoter chaque gène. Une présentation récente et détaillée du processus d'annotation des génomes est proposée par Reeves et al. [58].

Au cours de l'annotation, déterminer la fonction des protéines produites par un génome est une étape cruciale. Or, à partir d'un gène, nous n'obtenons qu'une séquence de protéine et cette seule séquence ne nous donne pas d'information sur sa fonction. Néanmoins, certaines informations peuvent être inférées à partir de la séquence d'une protéine. En 1973, Anfinsen a montré qu'une protéine retrouvait sa fonction après avoir été dénaturée [5]. Il en conclut que la fonction d'une protéine, et par extension sa structure tridimentionnelle (3D), sont codées dans sa séquence. De plus, Chothia et Lesk ont montré que deux protéines possédant des séquences très similaires se replient de façon similaire [16]. En fait, il est très probable que cette similarité soit due à une relation d'*homologie*. Deux protéines sont dites *homologues* si elles descendent d'un ancêtre commun. Cet ancêtre commun possèdait une séquence, une structure et une fonction particulière. Ses descendants peuvent avoir gardé ces caractéristiques. Deux protéines homologues peuvent donc avoir une même fonction, une structure 3D similaire et des séquences très proches. Par comparaison de séquences, nous pouvons donc inférer une relation d'homologie entre les protéines et en déduire la structure 3D (si celle d'un des homologues est connue), voire la fonction.

D'un point de vue informatique, une séquence de protéine est une chaîne de caractères comme une autre. La comparaison de chaînes de caractères est très utilisée en informatique et de nombreuses méthodes existent. En particulier, la distance de Levenshtein, ou distance d'édition, permet de mesurer la similarité entre deux chaînes de caractères. La distance de Levenshtein est directement applicable aux séquences de protéines. Aligner une séquence a avec une séquence b revient à transformer a en b à l'aide de trois opérations :

- la substitution, qui permet de remplacer un acide aminé par un autre ;
- l'insertion, qui permet d'insérer un acide aminé à une position ;
- la délétion, qui permet d'enlever un acide aminé à une position.

La définition originale de la distance de Levenshtein affecte un coût de 1 à ces opérations sauf si le caractère est identique. Dans le cas de séquences de protéines, des coûts plus élaborés ont été proposés [20, 32, 48, 19]. Ainsi, des séquences de protéines ont pu être comparées afin de détecter des homologies. Plusieurs méthodes ont été développées implémentant différents algorithmes (exacts ou non). Les méthodes Smith-Watermann [66], Needleman-Wunsch [50], Blast [2], PsiBlast [3] sont parmi les plus connues et les plus utilisées. Des études ont montré que des séquences de longueur supérieure à 100 acides aminés, et partageant plus de 30% d'identité de séquence, sont très probablement homologues [10, 62].

Cependant, en-dessous de 30% d'identité de séquence, les relations d'homologies deviennent difficiles à déterminer. Cette zone de flou est communément appelée *zone d'ombre* [60]. Brenner et al. ont montré que les méthodes par comparaison de séquences n'arrivaient à détecter que la moitié des protéines homologues ayant entre 20% et 30% d'identité de séquence [10]. En effet, dans cette zone, des protéines homologues et non-homologues présentent des taux d'identité de séquences similaires. Il devient donc difficile d'identifier clairement des relation d'homologies entre protéines.

La reconnaissance de repliements

Afin de détecter des protéines homologues dans la zone d'ombre, les méthodes dites de reconnaissance de repliements, ou *protein threading* en anglais, ont été développées dès 1990 [31, 64]. Ces méthodes se basent sur la constatation que la structure des protéines est bien mieux conservée que leur séquence au cours de l'évolution [33]. En effet, comme nous venons de le voir, des protéines ayant peu d'identité de séquences peuvent descendre d'un ancêtre commun et avoir une structure similaire. Il peut donc y avoir de nombreuses mutations dans une séquence de protéine sans que sa structure, voire sa fonction, ne soit changée. Les méthodes de reconnaissance de repliements cherchent donc à utiliser de l'information structurelle lors d'un alignement entre deux protéines. Or pour obtenir de l'information structurelle, il faut que la structure d'une des deux protéines soit connue.

Une question se pose alors : y a-t-il suffisamment de structures disponibles pour reconnaître n'importe quelle nouvelle séquence de protéine ? Zhang et al. ont montré que les structures disponibles dans la PDB (Protein Data Bank) suffisent à reconnaître quasiment n'importe quelle protéine mono-domaine [80]. De plus, de nombreuses études ont également montré que le nombre de repliements existants serait compris entre 1000 et 5000 [15, 70, 74, 18, 42]. De ces deux observations, nous pouvons en déduire que l'utilisation des structures résolues expérimentalement devrait nous permettre de classer la majorité des nouvelles protéines dans une famille existante. La réalité n'est, bien sûr, pas aussi simple.

Objectifs de cette thèse

Mon travail de thèse consiste essentiellement à proposer un nouveau type d'alignement pour les méthodes de reconnaissance de repliements.

Dans les méthodes d'alignements, l'utilisation des informations structurelles peut être locale ; l'information est portée par un acide aminé, une position. Un alignement optimal utilisant ce type d'information peut-être déterminé en un temps polynomial [66, 50]. Mais l'information structurelle peut également être portée par des paires d'acides aminés distants dans la séquence (*pairwise positions* en anglais). L'utilisation de ces informations pairées entraîne deux conséquences majeures :

1. une amélioration du taux de détection [77, 4, 17] ;
2. un alignement optimal bien plus complexe à déterminer [39].

En effet, Lathrop a montré que trouver un alignement optimal en utilisant des informations pairées, appelé *Protein Threading Problem* (PTP), est un problème NP-complet [39]. Le premier algorithme permettant de résoudre le PTP a été proposé par Lathrop et Smith en 1996 [40]. Cet algorithme permet de réaliser des alignements globaux dans lesquels tous les éléments de la structure sont obligatoirement alignés sur la séquence.

Dans le domaine des alignements de séquences, les méthodes les plus utilisées sont celles proposant des alignements locaux (Blast). Ces méthodes utilisent les trois opérations de transformation (substitution, insertion, délétion). Comme nous venons de le voir, la définition du PTP de Lathrop n'autorise pas les délétions dans la structure, c'est pourquoi nous parlons d'alignement global. À notre connaissance, il n'existe aucune méthode permettant de réaliser des alignements utilisant des informations structurelles pairées et autorisant les délétions dans la structure. Nous proposons donc une telle méthode d'alignement, dite locale, qui permettra de détecter des similarités locales entre protéines.

Ce travail de thèse est découpé en plusieurs étapes :

- formaliser le concept d'alignement local entre une séquence et une structure ;

- modéliser un alignement local sous la forme d'un problème d'optimisation ;
- trouver le meilleur modèle mathématique permettant de résoudre ce problème ;
- évaluer la pertinence des alignements locaux proposés ;
- résoudre le problème efficacement.

Plan de la thèse

Chapitre 1 - Le chapitre 1 est consacré à la prédiction de la structure des protéines. Après une introduction sur la constitution des protéines et de leurs structures, nous présentons un panorama des différentes méthodes permettant de prédire la structure d'une protéine. Nous nous intéressons plus particulièrement à la reconnaissance de repliements, domaine dans lequel s'inscrit ce travail de thèse.

Chapitre 2 - Dans ce chapitre, après une introduction à la programmation linéaire en nombres entiers (PLNE), nous présentons la modélisation du PTP proposée par Andonov et al. [4] ainsi que les méthodes utilisées pour le résoudre. Ce modèle et les notations mathématiques utilisées serviront de base pour le développement de nos modèles.

Chapitre 3 - Ce chapitre est consacré à la présentation de notre outil : FROSTO. Ce logiciel, basé sur le travail d'Antoine Marin et al. (FROST [44]), permet de réaliser des alignements entre protéines. Nous nous intéresserons en particulier aux fonctions de scores disponibles ainsi qu'aux améliorations obtenus grâce à la paralléllisation du code.

Chapitre 4 - Dans le chapitre 4, nous présentons notre première contribution : une extension du PTP permettant de réaliser un alignement local d'une structure avec une séquence. Nous proposons plusieurs modélisations MIP réalisées avec le logiciel CPLEX. Ces modélisations sont ensuite comparées en termes de temps de calcul et de qualité de modèle.

Chapitre 5 - Le chapitre 5 présente notre deuxième contribution : un algorithme dédié permettant de résoudre un alignement local tel que nous l'avons défini dans le chapitre 4. Cet algorithme utilise les techniques de relaxation Lagrangienne, de descente de sous-gradient et de résolution par séparation–évaluation (*Branch & Bound*) présentées dans le chapitre 4.

Chapitre 6 - Dans le chapitre 6, nous observons le comportement des alignements locaux. En particulier, nous montrons que notre méthode permet d'obtenir des alignements pour des cas où ce n'était pas possible via des alignements globaux. Nous abordons également les limitations liées aux fonctions de score et à la représentation des structures de protéines.

Chapitre 1

La prédiction de la structure des protéines

1.1 Les protéines

1.1.1 Définitions

La découverte des protéines est due au chimiste Gerrit Mulder (XIX^{me} siècle) mais ce n'est qu'entre 1939 et 1941, grâce aux travaux de Linus Pauling, que la structure des protéines commença à être réellement élucidée. Les protéines sont des macromolécules impliquées dans la majorité des réactions biologiques. Elles sont constituées d'acides aminés liés entre eux par des liaisons covalentes dites peptidiques. Les acides aminés possèdent une structure chimique commune constituée d'un carbone-α central lié à quatre groupes chimiques (figure 1.1) : un groupe amine (-NH_2), un groupe carboxyle (-COOH), un hydrogène (H) et une chaîne latérale (R). Il existe 20 acides aminés ayant des chaînes latérales différentes et présentant des caractéristiques physico-chimiques distinctes illustrées dans le diagramme de Venn de la figure 1.2.

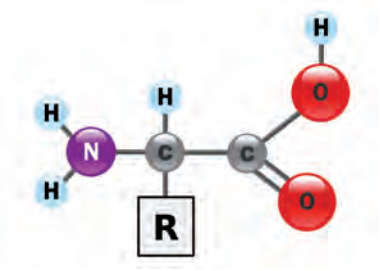

FIG. 1.1 – **Structure d'un acide aminé**. Un acide aminé est constitué d'un carbone-α central lié à quatre groupes : un groupe amine (-NH_2), un groupe carboxyle (-COOH), un hydrogène (H) et une chaîne latérale (R).

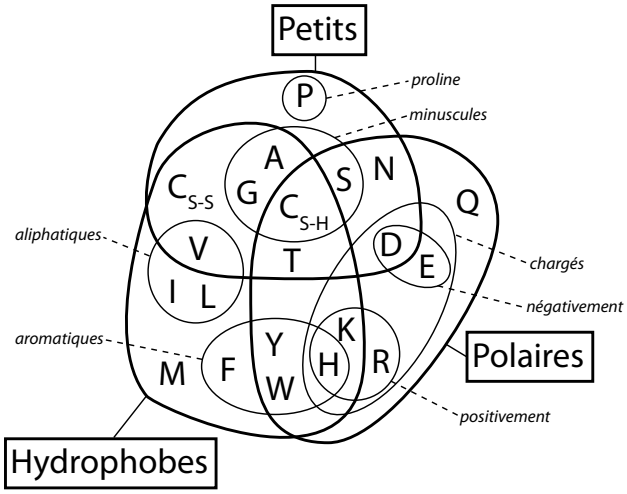

FIG. 1.2 – **Diagramme de Venn des caractéristiques physico-chimiques des acides aminés**.

L'enchaînement des acides aminés, également appelé *séquence* ou *chaîne polypeptidique*, est déterminé, codé, par un gène. Le passage d'un gène à une protéine s'effectue en deux étapes : la transcription puis la traduction. La transcription consiste à transcrire le code génétique en ARN messager qui sera ensuite traduit en protéine.

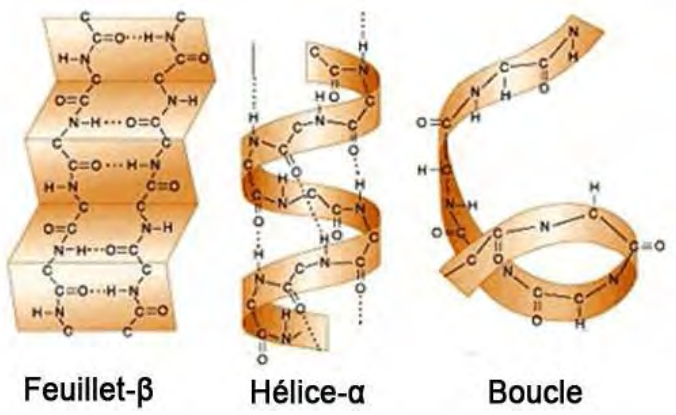

FIG. 1.3 – **Éléments de la structure secondaire des protéines**. Les hélices-α et les feuillets-β sont des motifs périodiques. Les boucles correspondent aux zones sans structure périodique.

Un système de description de la structure des protéines, organisé en quatre niveaux, a été proposé par Linderstrom-Lang en 1951. Le premier niveau, appelé *structure primaire*, correspond à l'enchaînement des acides aminés, la séquence. Le deuxième niveau, la *structure secondaire*, est une description basée sur l'organisation locale de la chaîne

polypeptidique, c'est-à-dire l'enchaînement des hélices-α, des feuillets-β et des boucles (figure 1.3). La structure tridimensionnelle de la protéine, c'est-à-dire la position de tous ses atomes dans l'espace, est appelée la *structure tertiaire*. C'est ce niveau de description qui sera considéré lorsque nous utiliserons les termes *structure* et *repliements* dans le reste du manuscrit. Enfin, certaines protéines forment des *complexes* protéiques qui correspondent à la *structure quaternaire*.

Les structures des protéines déterminées expérimentalement sont disponibles dans la banque de données PDB[1] (*Protein Data Bank*). À partir de la PDB, plusieurs classifications des structures des protéines ont été proposées. Elles consistent à regrouper les protéines par caractéristiques communes (fonction, structure, ou séquence) par des méthodes automatisées, comme CATH[2] [52], ou manuelles, comme SCOP[3] [49].

1.1.2 Séquençage et protéines

Grâce aux récents progrès technologiques et à l'arrivée des séquenceurs de nouvelle génération (NGS), la quantité de données génomiques croît exponentiellement. À partir de ces données, il est possible de déduire la séquence d'un grand nombre de protéines dont nous ne connaissons ni la structure, ni la fonction.

La structure des protéines est généralement déterminée par cristallographie ou par résonnance magnétique nucléaire. Néanmoins, ces deux méthodes sont coûteuse et surtout, elles sont beaucoup plus lentes que les technologies de séquençage.

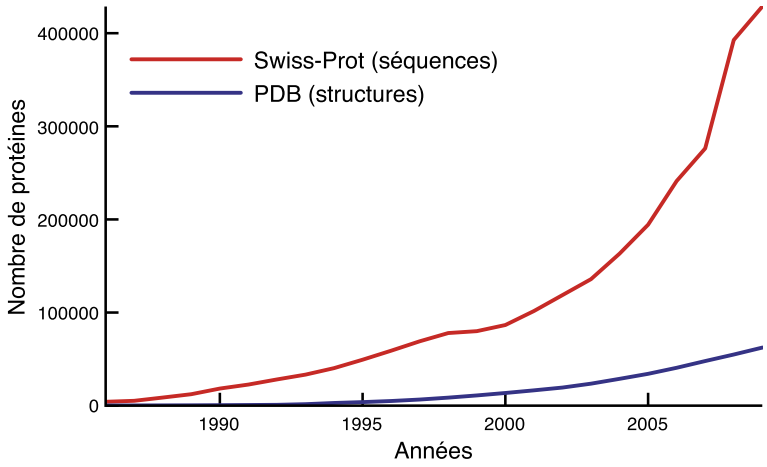

FIG. 1.4 – **Progression du nombre de séquences de protéines dans la banque de données Swiss-Prot et du nombre de structures de protéines dans la banque de données PDB entre 1986 et 2009**.

La figure 1.4 illustre la différence entre le nombre de structures disponibles dans

[1] http ://www.pdb.org
[2] http ://www.cathdb.info
[3] http ://scop.berkeley.edu

la PDB et le nombre de séquences disponibles dans Swiss-Prot[4] (base de séquences de protéines annotées). Nous pouvons voir que le nombre de séquences annotées croît exponentiellement. Cependant, la différence entre le nombre de structures et le nombre de séquences disponibles est bien plus grand car le nombre total de séquences (annotées ou non) se compte en millions.

Il existe donc, d'une part, un besoin important et toujours grandissant de connaître la fonction et éventuellement la structure des protéines dans le processus d'annotation, et d'autre part, des méthodes expérimentales qui ne peuvent répondre à cette demande. C'est à cette étape qu'interviennent les méthodes bioinformatiques de prédiction de la structure des protéines.

1.2 La prédiction de la structure des protéines

Comment déterminer la structure d'une protéine à partir de sa seule séquence ? Pour répondre à cette question, deux approches différentes sont couramment utilisées. La première approche consiste à construire un modèle 3D d'une protéine directement à partir des acides aminés de sa séquence. les méthodes utilisant cette approche sont dites *ab initio*. La deuxième approche consiste à essayer de trouver des protéines "similaires" dans des banques de protéines connues. À partir de ces protéines, un modèle 3D de la structure est construit. Ce travail de thèse se situant dans la deuxième approche, nous ne détaillerons pas les méthodes *ab initio*. Néanmoins, le lecteur intéressé pourra se reporter à la comparaison récente des méthodes *ab initio* dans le numéro spécial de *Proteins* dédié à la compétition CASP 8 [7].

Les méthodes utilisant la deuxième approche, c'est-à-dire cherchant des "similarités" entre les séquences, se basent sur le principe suivant :

- Si deux protéines partagent une forte similarité de séquence, il est très probable que cette similarité soit due à une relation d'*homologie* [62].

Ce principe permet d'inférer des connaissances sur une protéine en cherchant des homologues à celle-ci dans les bases de données de protéines connues. Deux protéines sont homologues si elles partagent un ancêtre commun. Elles peuvent avoir gardé une fonction, une structure et une séquence similaires à celles de cet ancêtre. Les méthodes basées sur ce principe, dites méthodes par homologie, se déroulent généralement en trois étapes :

1. des homologues sont recherchés dans des bases de données ;
2. si la structure 3D d'un des homologues détectés est connue, un modèle 3D est construit ;
3. la qualité du modèle 3D est évaluée afin de proposer des raffinements.

[4] http ://www.expasy.ch/sprot/

La première étape est évidemment capitale dans la suite du processus. Plus nous trouvons d'homologues proches dont on connaît la structure, plus la création du modèle 3D sera aisée et précise. Mon travail de thèse se situe dans cette première étape. Nous proposons une nouvelle méthode de recherche d'homologues. Les étapes deux et trois étant hors du cadre de ce travail, nous ne les présenterons pas. Néanmoins, le lecteur intéressé pourra se reporter au numéro spécial de *Proteins* consacré aux résultats de la compétition CASP 8 dont une partie traite de la construction de modèles 3D et de leur évaluation [47].

Dans la section suivante, nous présentons un panorama des méthodes de recherche d'homologues.

1.3 Les méthodes de détection par homologie

L'objet de cette section est de présenter les approches permettant de rechercher des protéines homologues. Devant le grand nombre de méthodes disponibles, nous ne pouvons toutes les détailler. Nous présentons ici les différentes approches en fonction des données utilisées et des types d'algorithmes d'alignement proposés.

1.3.1 La similarité de séquences

Lorsqu'une séquence de protéine est déterminée, la première étape est de la comparer aux séquences de protéines contenues dans les bases de données publiques. Si une similarité suffisante est trouvée entre deux protéines, elles ont alors une forte probabilité d'être homologues [62] et donc de partager une structure et une fonction similaires. La recherche de similarité entre deux séquences des protéines est un domaine très actif qui a donné naissance à une multitude de logiciels et de techniques. Nous proposons de les différencier par le type d'alignements réalisés et par la mesure de similarité utilisée entre les séquences.

1.3.1.1 Les types d'alignements

La plupart des méthodes de comparaison de séquences se basent sur les techniques de programmation dynamique proposées par Needleman et Wunsch [50], puis par Smith et Waterman [66]. Ces techniques proposent d'aligner deux séquences de protéines en prenant les acides aminés comme éléments unitaires. Un acide aminé de la protéine A peut-être aligné face à un acide aminé de la protéine B ou face à un "trou" (*gap* en anglais). Selon la fonction de score utilisée, un score est affecté à l'alignement de deux acides aminés. De la même façon, un score est affecté à l'alignement d'un acide aminé avec un *gap*.

À partir de ce schéma général, nous pouvons définir trois types d'alignements en fonction de nos besoins (figure 1.5) :

- l'alignement global ;
- l'alignement semi-global ;
- l'alignement local.

Un alignement global est utilisé lorsque nous souhaitons détecter des similarités entre protéines d'une même famille. De telles protéines sont censées avoir des longueurs similaires. En conséquence, les gaps aux extrémités seront pénalisés.

Néanmoins, de nombreuses protéines ont une structure modulaire, c'est-à-dire constituée de plusieurs modules, ou domaines. De plus, des protéines différentes peuvent partager un même domaine. Lorsque nous souhaitons détecter un domaine, autrement dit une petite séquence à l'intérieur d'une grande séquence, nous devons utiliser un alignement semi-global. Dans ce type d'alignement, les gaps aux extrémités ne sont pas pénalisés afin de permettre au domaine de s'aligner au mieux sur la grande séquence. Par contre, les gaps à l'intérieur des séquences sont toujours pénalisés.

L'alignement local est le plus général. Il consiste à détecter des similarité locales entre deux séquences, par exemple détecter un domaine commun à deux protéines. C'est le type d'alignement le plus utilisé, en particulier dans le logiciel BLAST.

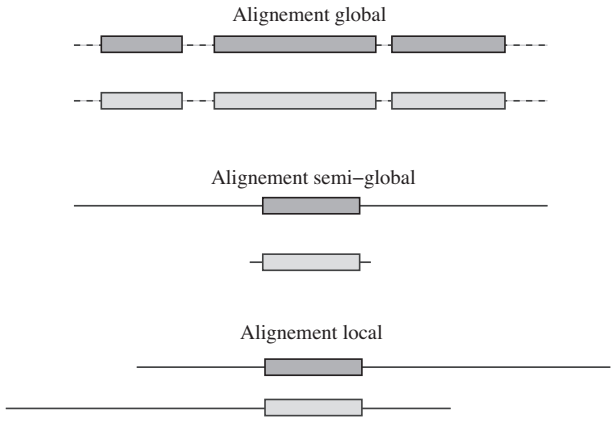

FIG. 1.5 – **Types d'alignement**. Lors d'une comparaison de séquences de protéines, il est important de pouvoir réaliser différents types d'alignement afin de s'adapter à la situation. **Alignement global :** ce type d'alignement est utilisé lorsque nous souhaitons comparer des protéines d'une même famille. **Alignement semi-global :** Un alignement semi-global est utilisé lorsque nous souhaitons détecter un domaine dans une séquence de protéine. Les gaps aux extrémités ne sont plus pénalisés. **Alignement local :** C'est le type d'alignement le plus général. Il est utilisé pour détecter des similarités locales entre deux séquences de protéines. Par exemple détecter un domaine commun.

Ces différents types d'alignements peuvent être obtenus par un algorithme de programmation dynamique. Cette technique permet d'obtenir l'alignement optimal entre deux séquences en un temps polynomial. Néanmoins, la taille des bases de données croit exponentiellement et bien plus vite que la puissance des processeurs. En conséquence, chercher l'alignement optimal entre deux séquences devient très coûteux sur une telle quantité de données. Commme la méthode BLAST [2] est basée sur une heuristique permettant de trouver des alignements locaux sub-optimaux très rapidement, elle est souvent préférée à des méthodes d'alignement exactes.

Afin d'ajouter de l'information à une séquence de protéine, des méthodes se sont intéressées à aligner des séquences d'une même famille. Ces alignements, dits multiples (ou *MSA* pour Multiple Sequence Alignment), permettent d'observer les variations des acides aminés à une position donnée dans des familles de protéines. Les MSAs servent en particulier à calculer des fréquences d'apparition des acides aminés à certaines positions afin d'en définir le profil statistique. Parmi les logiciels d'alignements multiples les plus utilisés, nous pouvons citer : ClustalW [68], MUSCLE [23] et DIALIGN [46].

1.3.1.2 Les mesures de similarité

La similarité entre deux séquences peut se définir de multiples façons. La plus simple est de considérer l'identité stricte entre les acides aminés. Dans ce cas, aligner deux acides aminés identiques vaut 1 ou 0 autrement. Néanmoins, nous savons que certaines substitutions peuvent avoir lieu sans changer la structure ou la fonction de la protéine.

Afin de mesurer les coûts de substitutions entre acides aminés, Dayhoff et al. proposèrent, en 1978, d'étudier la "mutabilité" des acides aminés [20]. Les résultats de cette étude sont les matrices de substitutions PAM qui ont été largement utilisées pour calculer des similarités entre séquences de protéines. De nombreuses matrices de substitutions ont depuis été proposées, comme les matrices BLOSUM [32, 48, 19] basées sur la fréquence des acides aminés dans des alignements multiples. Les coûts de substitution de ces matrices sont calculés sur un grand nombre de protéines et représentent donc des coûts moyens.

Afin d'augmenter l'information portée par une position dans une séquence, il est également possible d'utiliser un alignement multiple. À chaque position d'un MSA, la fréquence de chaque acide aminé observé peut être calculée produisant ainsi un profil statistique de la position. Plus l'alignement multiple est précis, plus les scores extraits de cet alignement sont sensibles et permettent de détecter des similarités [53]. Afin de les raffiner, les MSAs générés automatiquement peuvent être modifiés par des experts afin de corriger certaines erreurs [26]. À partir d'un MSA, nous pouvons donc générer des profils spécifiques à une famille de protéines [54], mais également des Modèles de Marokov Cachés (HMM) [73]. Ces profils sont utilisés dans des fonctions de score et ont donné lieu à des alignements profil–séquence [63], HMM–séquence [22], profil–profil [61], ou HMM–HMM [67] qui permettent de détecter des homologues plus distants [21, 34].

Les scores de substitution ne sont pas les seuls scores qui participent à la mesure de similarité. En effet, le coût des gaps influe fortement sur la qualité des alignements et donc sur la qualité de la détection d'homologues [57, 27, 78].

1.3.1.3 La zone d'ombre

Les méthodes d'alignement de séquences ont montré leur efficacité [56]. Néanmoins, en-dessous de 30% d'identité de séquence, des protéines homologues et non-homologues peuvent avoir des taux de similarité de séquences identiques. De plus, Brenner et al. montrent que, sur un jeu de 9044 paires de protéines homologues partageant moins de 40% d'identité de séquences, seules 18% sont reconnues par des méthodes d'alignements de séquences [10]. Dans cette *zone d'ombre*, les méthodes d'alignement de séquences ne sont donc plus suffisamment puissantes pour détecter des homologies entre protéines [60].

Afin de détecter des homologies dans la *zone d'ombre*, des méthodes, dites de reconnaissance de repliements, ont été développées. Ces méthodes utilisent le fait que la structure des protéines est mieux conservée que leur séquence au cours de l'évolution [33].

1.3.2 Reconnaissance de repliements

Les méthodes de reconnaissance de repliements, ou "*Protein Threading*" en anglais, consistent à mesurer la compatibilité entre une séquence et une structure de protéine [9]. Toutes ces méthodes comprennent les quatres éléments suivants :

- une représentation de la structure des protéines ;
- une fonction de score évaluant la compatibilité entre une structure et une séquence ;
- un algorithme d'alignement (optimal ou approché) ;
- une statistique permettant d'évaluer la significativité des alignements.

La représentation de la structure des protéine peut prendre en compte de nombreuses informations structurelles En effet, un acide aminé peut être enfoui au coeur de la protéine ou exposé au solvant ; être dans une hélice, un feuillet ou une boucle ; être en contact avec d'autres acides aminés ; faire partie du site actif de la protéine ; former un pont disulfure (cystéine) ; etc. Ces informations sont directement extraites des fichiers PDB.

Un fichier PDB est issu de la plus importante banque publique de structures de protéines : la Protein Data Bank (PDB). Ces fichiers de la PDB décrivent les coordonnées 3D des atomes des protéines déterminées expérimentalement par cristallographie ou par résonnance magnétique nucléaire. La qualité des informations structurelles extraitent de ces fichiers dépend évidemment de la qualité des expérimentations qui ont été réalisées.

Pour les séquences, dont on ne connaît pas la structure, certaines informations structurelles peuvent être prédites. De nombreuses méthodes proposent de prédire des structures locales comme les régions coiled-coil [43], les hélices transmembranaires [37], les éléments de structure secondaire [35], les zones désordonnées [71] ou de faible complexité [75]. Ces prédictions ont été utilisées, par exemple, pour enrichir des matrices de substitutions [59, 13].

L'utilisation des informations structurelles (prédites ou observées), dans une fonction de score, peut être locale ou non. Une information structurelle locale, par exemple l'exposition au solvant d'un acide aminé, sera associée à une position. Au lieu de considérer seulement un acide aminé, nous considérons cet acide aminé dans un certain état structurel. Les profils (statistique ou HMM) sont enrichis par ces informations plus spécifiques. L'utilisation du contexte structurel des acides aminés, qu'il soit prédit ou observé, a permis d'améliorer la détection d'homologues dans la zone d'ombre [34]. La recherche d'un alignement optimal utilisant des informations locales se résout par un algorithme classique de programmation dynamique en un temps polynomial [50, 66]. Néanmoins, les informations locales ne reflètent pas la structure tertiaire des protéines.

Afin de prendre en compte la structure tertiaire, l'information structurelle n'est plus associée à un seul acide aminé mais à une paire d'acides aminés proches dans l'espace. Deux acides aminés sont dits "en contact", ou "en interaction", si la distance entre leurs carbones-α est inférieure à un certain seuil. L'information portée par les paires d'acides aminés est dite "pairée". Les informations structurelles pairées peuvent être de nature physique [65, 11] ou de nature statistique [38, 76, 79]. L'utilisation de l'information structurelle pairée a permis d'augmenter le taux de reconnaissance de repliements dans la zone d'ombre [40, 44, 77, 79].

Cependant, trouver un alignement optimal en utilisant des informations pairées est un problème NP-Complet [39], et même Max SNP-Difficile [1]. Pour résoudre un tel alignement, des méthodes utilisant la programmation linéaire en nombres entiers (PLNE) ont été développées sur la base du problème du "*Protein Threading*" [39].

1.3.3 Le problème du "*Protein Threading*"

Dans cette section, nous présentons la formulation du problème du "*Protein Threading*", ou PTP, qui a été proposé en 1994 par Lathrop [39]. Bien qu'utilisant le terme de "*Protein Threading*", le PTP est un problème d'alignement particulier dans le domaine de la reconnaissance de repliements. Le PTP consiste à aligner une structure de protéine avec une séquence requête en utilisant des information pairées.

Représentation de la séquence requête La séquence requête est représentée par l'enchaînement de ses acides aminés. D'un point de vue informatique, il s'agit d'une chaîne de caractères sur un alphabet de 20 lettres. C'est la protéine de structure inconnue

qui sera alignée contre les structures d'une base de données. Les notations associées à la séquence requête sont les suivantes :

- la séquence requête est noté S ;
- le nombre de résidus dans S, sa longueur, est noté N.

Représentation des structures de protéines Les méthodes de reconnaissance de repliements n'utilisent pas directement les coordonnées 3D des atomes des structures des protéines. Les structures sont définies en terme de blocs d'acides aminés, d'environnement structurel, de distances, de contacts, etc (figure 1.6). Dans le PTP, les blocs d'acides aminés correspondent aux parties les plus conservées des structures dans lesquelles aucun *gap* ne sera admis. Classiquement, les blocs correspondent aux éléments de structure secondaires (SSEs : hélice-α et feuillets-β). Afin de formaliser la représentation des structures, nous introduisons les notations suivantes :

- M est l'ensemble ordonné des m blocs de la structure ($|M| = m$) ;
- un bloc k possède une longueur fixe L_k (le nombre d'acides aminés dans le bloc) ;
- $I \subseteq \{(k,l) \mid 1 \leq k < l \leq m\}$ est l'ensemble des interactions entre blocs. Il existe une interaction entre les blocs k et l si au moins un acide aminé du bloc k est en contact avec un acide aminé du bloc l ;
- une structure est représentée par le graphe $G(M, I)$ appelé *carte de contacts généralisée*.

Alignement Dans le PTP classique, un alignement correspond à assigner les acides aminés de la séquence dans les blocs de la structure. Comme indiqué dans la figure 1.7, tous les acides aminés de la séquence ne sont pas forcément assignés. Par contre, tous les blocs de la structure doivent être alignés sur la séquence. Cette propriété n'est pas commune à la définition traditionnelle d'un alignement. En effet, dans un alignement de séquence, les insertions, délétions et substitutions sont possibles dans les deux séquences alignées. Dans le PTP, les blocs de la structure sont obligatoirement alignés. En conséquence, un alignement correspond à positionner les blocs le long de la séquence. Un tel alignement est dit *faisable*, si les blocs restent dans l'ordre et ne se chevauchent pas. Un alignement est donc entièrement décrit par la position absolue du premier acide aminé de chaque bloc sur la séquence.

Cependant, dans la formulation classique du PTP, il est plus aisé d'utiliser des positions relatives plutôt qu'absolues. Si le bloc k est placé en position absolue i sur la séquence, alors sa position relative est $r_{ik} = i - \sum_{j=1}^{k-1} L_j$. Ainsi, tous les blocs ont le même nombre de positions relatives possibles $n = N - \sum_{k=1}^{m} L_k + 1$. Un exemple de correspondance entre les positions absolues et relatives est donné dans le tableau 1.1.

L'ensemble des alignements faisables, c'est-à-dire respectant les contraintes d'ordre et de non chevauchement des blocs, est noté $\mathcal{T} = \{(r_1, \ldots, r_m) \mid 1 \leq r_1 \leq \cdots \leq r_m \leq n\}$. La taille de cet ensemble, c'est-à-dire la taille de l'espace des solutions, est donnée par $|\mathcal{T}| = \binom{m+n-1}{m}$. Le nombre d'alignements possible est donc très grand même pour des

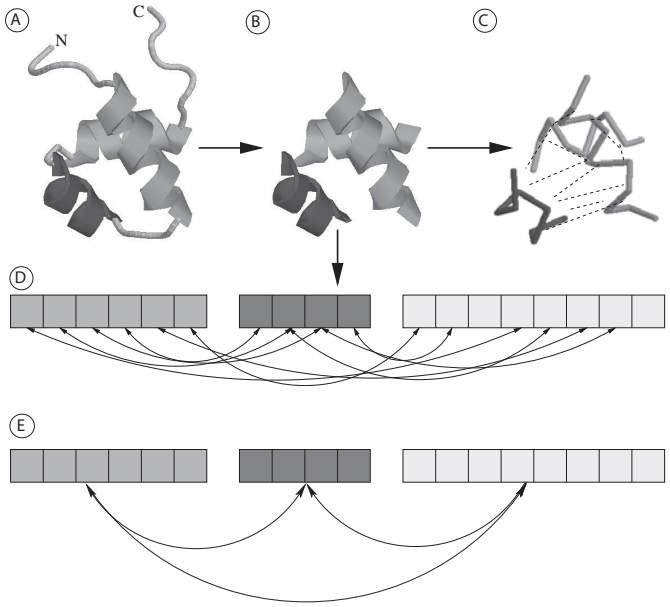

FIG. 1.6 – **Transformation d'une structure de protéine en une carte de contacts généralisée**. **A)** Représentation "en ruban" d'une structure 3D composée de 3 hélices-α. **B)** Seuls les hélices sont conservées, chacune définissant un bloc. **C)** Les blocs sont réduits à la position de leurs carbones-α dans l'espace. Les carbones-α sont reliés par un trait épais. Les lignes pointillées représentent les interactions entre acides aminés. Deux acides aminés sont en interaction si la distance entre leur carbones-α est inférieure à un seuil donné. **D)** Carte de contacts de la structure. Les cases représentent les acides aminés des blocs. Les arcs fléchés représentent les lignes pointillées de C, c'est-à-dire les interactions entre acides aminés. **E)** Carte de contacts généralisée de la structure définie par le graphe $G = (M, I)$ avec $M = \{1, 2, 3\}$ et $I = \{(1, 2), (1, 3), (2, 3)\}$. Un arc fléché est tracé entre deux blocs s'il existe au moins une interaction entre ces deux blocs.

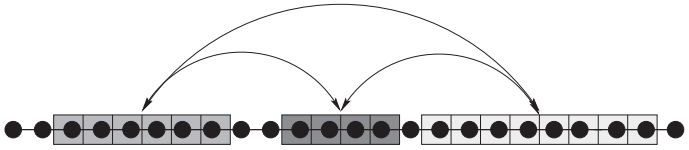

FIG. 1.7 – **Exemple d'alignement**. La carte de contact généralisée de la figure 1.6 est alignée avec une séquence de 24 acides aminés. L'alignement peut-être défini par les positions absolues des premiers acides aminés de chaque bloc : (3,11,16). Le score d'un tel alignement est la somme des scores de toutes les interactions entre blocs, c'est-à-dire de toutes les paires de positions où les acides aminés en contact.

abs. position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
rel. position block 1	1	2	**3**	4	5	6	7																	
rel. position block 2							1	2	3	4	**5**	6	7											
rel. position block 3											1	2	3	4	5	**6**	7							

TAB. 1.1 – **Correspondance entre positions absolues et positions relatives**. Les positions relatives en gras correspondent aux positions des blocs dans l'alignement de la figure 1.7. Ainsi, si le bloc 1 est en position relative 7, les blocs 2 et 3 sont forcément en position relative 7. Cela signifie que tous les blocs sont placés le plus à droite possible sur la séquence.

problèmes de taille moyenne (si $m = 20$ et $n = 100$, alors $|\mathcal{T}| \approx 2.5 \times 10^{22}$).

Le PTP ainsi formulé peut être résolu par programmation linéaire en nombres entiers (PLNE). Dans la section suivante, après une présentation générale de la PLNE, nous présentons une modélisation du PTP, appelée modèle MXY, développée par Andonov et al. [4]. Nous présentons également les méthodes permettant de le résoudre efficacement.

1.4 Ce qu'il faut retenir de ce chapitre

Afin de prédire la structure d'un protéine, deux voies sont possibles. Si une homologie avec une ou plusieurs protéines connues est détectée, ces protéines serviront de base à la fabrication d'un modèle 3D. Si aucune homologie n'est détectée, les méthodes dites *ab initio* [7] sont utilisées pour construire un modèle 3D.

Afin de détecter une homologie, la première étape consiste à aligner une séquence de protéine de structure inconnue avec des séquences de protéines connues. Si une forte similarité est détectée, il est très probable que celle-ci soit due à une relation d'homologie [62]. Néanmoins, en-dessous de 30% d'identité de séquence, les mesures de similarité de séquences ne sont plus suffisante pour détecter des homologies [10, 60]. Cette zone est appelée *zone d'ombre*.

La reconnaissance de repliements consiste à ajouter de l'information structurelle dans les alignements de séquences afin de détecter des homologies dans la zone d'ombre. Les méthodes de reconnaissance de repliements comprennent les quatre éléments suivants :

- une représentation des protéines ;
- une fonction évaluant la compatibilité entre une structure et une séquence ;
- un algorithme d'alignement (optimal ou approché) ;
- une statistique permettant d'évaluer la significativité des alignements.

Représentation des protéines Les représentations des protéines couramment utilisées diffèrent par l'élément unitaire aligné et le type d'information structurelle utilisée.

L'élément unitaire peut être soit l'acide aminé soit un bloc d'acides aminés. Si l'information structurelle est locale, alors la protéine est représentée par la chaîne d'éléments unitaires. Si l'information structurelle est pairée, la protéine est représentée par une carte de contacts, c'est-à-dire un graphe orienté dont les sommets sont les éléments unitaires et dont les arcs représentent les contacts entre les éléments unitaires (figure 1.6).

Évaluation de la compatibilité entre une structure et une séquence La compatibilité entre une séquence et une structure peut être évaluée par une fonction de score prenant en compte l'environnement structurel local des éléments unitaires (acides aminés ou blocs). La fonction de score peut également prendre en compte des interactions distantes entre acides aminés mais cela transforme la recherche d'un alignement optimal en problème NP-Complet [39].

Algorithme d'alignement Si l'information structurelle utilisée est locale, la recherche de l'alignement optimal entre une séquence et une structure peut être résolu en un temps polynomial par un algorithme de programmation dynamique [50, 66]. Afin de prendre en compte l'information structurelle distante, des algorithmes par programmation linéaire en nombres entiers ont été développés [40, 77, 4]. Néanmoins, étant basés sur le PTP, ces algorithmes ne permettent pas les délétions dans la structure alignée. Dans ce manuscrit, nous proposons une méthode d'alignement utilisant des informations structurelles distantes et permettant les délétions dans la structure.

Évaluation et tri des alignements Lorsqu'une séquence de protéine est comparée à une banque de données, nous avons besoin de trier les résultats du plus au moins similaire. Ceci peut-être réalisé directement si le score d'alignement le permet. Néanmoins, dans la plupart des cas, comparer le score brut de deux alignements n'est pas significatif. Des méthodes statistiques sont alors nécessaires afin de mesurer la significativité des alignements (p-value, z-scores, etc).

Chapitre 2

Modélisation d'alignements globaux par programmation linéaire en nombres entiers

Dans le chapitre précédent, nous avons décrit le concept du PTP. Ce problème consiste à trouver le positionnement optimal de blocs d'acides aminés le long d'une séquence. Il s'agit donc d'un problème d'optimisation qui peut être modélisé par programmation linéaire en nombres entiers (PLNE) ou *Mixed Integer Programming* (MIP) en anglais.

Dans ce chapitre, après la présentation d'un problème classique de sac-à-dos modélisé en PLNE, nous décrivons la modélisation du PTP proposée par Andonov et al. [4]. Ce modèle, ainsi que les notations utilisées, serviront de base pour la descriptions des modèles proposés dans ce travail de thèse. Nous abordons ensuite les méthodes permettant de résoudre ces modèles.

2.1 La programmation linéaire en nombres entiers

La programmation linéaire en nombres entiers est un domaine de la recherche opérationnelle qui consiste à décrire un problème d'optimisation sous une forme mathématique comme présenté ci-dessous. Cette description est appelée modèle MIP et pour un même problème, plusieurs modèles MIP peuvent être proposés.

Un modèle MIP est constitué de deux parties : une fonction objectif et des contraintes. La fonction objectif représente ce que nous souhaitons optimiser, il s'agit donc d'une minimisation ou d'une maximisation. De plus, nous voulons optimiser cette fonction sous certaines contraintes (d'espace, de temps, d'efficacité, etc). Dans un problème modélisé par PLNE, la fonction objectif et les contraintes sont linéaires. Nous proposons d'illustrer la conception d'un modèle MIP sur un problème classique : le problème du sac-à-dos.

Le problème est de remplir un sac-à-dos avec des objets plus ou moins utiles. Comme nous souhaitons emmener le plus d'objets utiles dans le sac-à-dos, il s'agit donc de maximiser "l'utilité totale" contenue dans le sac-à-dos. Néanmoins, les objets prennent une certaine place et nous ne pouvons pas tous les mettre dans le sac-à-dos. Des contraintes s'appliquent sur la maximisation. Le problème du sac-à-dos est ici limité à un seul objet de chaque type.

Objets	boussole	carte	sandwich	bouteille d'eau	canif	tournevis
Variable	X_1	X_2	X_3	X_4	X_5	X_6
Utilité	0.2	0.5	0.7	1.0	0.4	0.01
Encombrement	1	2	4	6	1	1

TAB. 2.1 – **Exemple de données pour un problème de sac-à-dos (0/1)**.

Afin de modéliser ce problème, nous utilisons des variables binaires. Soit X_i la variable binaire associée à l'objet i. Si X_i vaut 1 alors l'objet est mis dans le sac-à-dos, sinon X_i vaut 0. Soit U_i un nombre réel représentant l'utilité de l'objet i et n le nombre d'objets disponibles. La fonction objectif de notre problème est alors :

$$Max \sum_{i=1}^{n} U_i X_i \tag{2.1}$$

Dans cette équation, si X_i vaut 1, alors l'utilité U_i est ajoutée au score. Nous ajoutons donc bien l'utilité d'un objet lorsque nous le mettons dans le sac-à-dos.

Soit E_i l'encombrement de l'objet i et C la contenance totale du sac-à-dos. La contrainte d'encombrement du modèle est :

$$\sum_{i=1}^{n} E_i X_i \leq C \tag{2.2}$$

Dans cette équation, l'encombrement de l'objet i est pris en compte si la variable X_i vaut 1. Si la somme des encombrements des objets dépasse la valeur C alors tous les X_i ne pourront pas valoir 1 en même temps (tous les objets ne rentrent pas dans le sac). Cette formulation est un modèle MIP, c'est-à-dire une modélisation du problème en PLNE. Il s'agira ensuite de le résoudre, c'est-à-dire de trouver une solution acceptable pour les contraintes et qui maximise l'utilité.

Si nous appliquons les données du tableau 2.1 à un sac-à-dos de contenance $C = 12$, nous obtenons la fonction objectif suivante :

$$Max \;\; 0.2\,X_1 + 0.5\,X_2 + 0.7\,X_3 + 1.0\,X_4 + 0.4\,X_5 + 0.01\,X_6 \tag{2.3}$$

Cette fonction objective est soumise à la contrainte suivante :

$$1\,X_1 + 2\,X_2 + 4\,X_3 + 6\,X_4 + 1\,X_5 + 1\,X_6 \leq 12 \tag{2.4}$$

Dans ce modèle, la solution optimale est de prendre la bouteille d'eau, le sandwich, la boussole et le canif pour une utilité totale de 2.3 et un encombrement de 12. Nous pouvons trouver cette solution facilement car il n'y a qu'une contrainte et peu de variables. Néanmoins, lorsqu'un modèle contient plusieurs dizaines de milliers de variables et de contraintes, des méthodes de résolution efficaces sont nécessaires.

Le PTP est un problème qui ressemble beaucoup au sac à dos puisqu'il s'agit de positionner des blocs, ayant un score (utilité) et une longueur (encombrement), sur une séquence ayant une certaine longueur (contenance). La section suivante décrit la modélisation MIP du PTP proposée par Andonov et al. [4].

2.2 Formulation du PTP en modèle MIP

Afin de faciliter la modélisation du PTP en PLNE, Andonov et al. ont considéré que rechercher le meilleur alignement correspondait à rechercher le chemin optimal dans un graphe de flots. La section suivante présente la correspondance entre ces deux problèmes d'optimisation.

2.2.1 Modélisation en graphe de flots

Soit $G(V, E)$ un graphe orienté. L'ensemble des sommets V est organisé en grille de taille $n \times m$ comme illustré dans la figure 2.1 B. Un sommet ik de cette grille représente le bloc k assigné à la position relative i (définie dans la section 1.3.3). Dans cette section, lorsque nous utilisons le terme "position", nous entendons toujours "position relative".

Un sommet (i,k) de cette grille représente donc le bloc k à la position i sur la séquence. L'ensemble des arcs E représente les interactions entre blocs comme défini en section 1.3.3. Pour chaque paire de blocs en interaction $(k, l) \in I$ $(k < l)$, aux positions respectives i et j telles que $i \leq j$, il existe un arc $(i, k)(j, l)$ dans E. Cette définition des arcs garantit que les chemins possibles dans le graphe respectent les contraintes d'ordre et de non-chevauchement des blocs. Le graphe G ainsi défini est appelé *graphe d'alignement*.

Selon la fonction de score considérée, un score C_{ik} est associé à chaque sommet (i, k) et un score C_{ikjl} est associé à chaque arc $(i, k)(j, l)$. C_{ik} est un score local qui ne dépend que du block k à la position i. C_{ikjl} est un score non-local (*pairé*) qui dépend conjointement du block k à la position i et du bloc l à la position j.

Nous ajoutons à ce graphe un sommet de départ S, un sommet de fin T et les arcs correspondants $(S, (i, 1))$ et $((i, m), T)$, pour $i \in [1, n]$. Le graphe ainsi défini est un

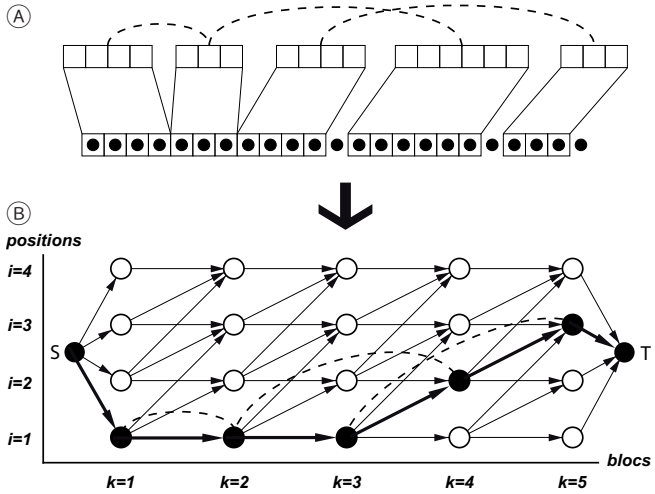

FIG. 2.1 – **Correspondance entre un PTP et un graphe de flots**. **A)** Exemple d'alignement d'une structure contenant 5 blocs sur une séquence de 23 acides aminés. Les arcs en pointillés représentent les interactions entre blocs. **B)** Graphe de flots correspondant au problème d'alignement de **A**. Les sommets en noir sont activés par le chemin représentant l'alignement de **A**. Les lignes pleines représentent les chemins possibles, les lignes pointillées représentent les interactions entre blocs. Pour des raisons de lisibilités, nous n'avons représenté que les interactions activées par le chemin.

graphe de flots. Un alignement d'une structure sur une séquence correspond exactement à un chemin, ou flot de volume 1, de S à T (figure 2.1). Nous disons qu'un chemin de S à T *active* les sommets qu'il traverse. Un arc est activé par un chemin si les sommets qu'il relie sont activés. Tout arc ou sommet activé par un chemin participe au score final. Le sous-graphe des sommets et des arcs activés par un chemin est appelé *chemin augmenté* dans le sens où il active des interactions distantes, c'est-à-dire des arcs entre des colonnes non-consécutives. Trouver le chemin augmenté optimal dans G est équivalent à trouver l'alignement optimal entre une structure et une séquence.

2.2.2 Modèlisation MIP : le modèle MYZ

À partir du graphe de flots défini dans la section précédente, Andonov et al. ont développé un modèle MIP appelé *MYZ*. Ce modèle a servi de base aux modèles développés au cours de ce travail de thèse, c'est pourquoi nous le détaillons ici afin d'introduire les notations et les concepts utilisés.

Pour plus de clarté, nous rappelons ici quelques notations présentées dans le chapitre précédent :

- N est la longueur de la séquence requête ;
- m est le nombre de blocs dans la structure ;
- L_k est la longueur du bloc k ;

- n est le nombre de positions relatives des blocs sur la séquence ;

Dans le graphe de flots, un chemin active les sommets et les arcs qu'il traverse. L'activation des sommets est représentée par un ensemble Y de variables binaires. À chaque sommet (i,k) est associé une variable binaire $y_{ik} \in Y$ qui vaut 1 si le bloc k à la position i est activé, et 0 sinon. Ces variables nous permettent de définir l'espace des chemins possibles par les contraintes suivantes :

$$Y_{ik} \in \{0,1\} \qquad k \in [1,m],\ i \in [1,n] \tag{2.5}$$

$$\sum_{i=1}^{n} Y_{ik} = 1, \qquad k \in [1,m] \tag{2.6}$$

$$\sum_{j=1}^{i} Y_{j(k+1)} - \sum_{j=1}^{i} Y_{jk} \leq 0 \qquad k \in [1,m-1],\ i \in [1,n-1] \tag{2.7}$$

Les contraintes (2.6) forcent l'activation d'une et une seule variable y dans chaque colonne. En effet, un bloc doit être aligné à une seule position. Les contraintes (2.7) garantissent que le chemin est croissant. Autrement dit, l'activation d'un sommet (i,k) force l'alignement du bloc suivant à une position $j \geq i$.

L'activation des arcs est représentée par un ensemble Z de variables réelles. À chaque arc $(i,k),(j,l)$ est associée une variable $z_{ikjl} \in Z$ qui vaut 1 si les blocs k et l aux positions i et j sont activés. Autrement dit, un arc est activé si les sommets à ses extrémités sont activés. Ce comportement est représenté par les contraintes (2.8) et (2.9) :

$$Y_{jl} = \sum_{i=1}^{j} Z_{ikjl} \qquad (k,l) \in I,\ j \in [1,n] \tag{2.8}$$

$$Y_{ik} = \sum_{j=i}^{n} Z_{ikjl} \qquad (k,l) \in I,\ i \in [1,n] \tag{2.9}$$

$$0 \leq Z_{ikjl} \leq 1 \qquad (i,k,j,l) \in E \tag{2.10}$$

Les variables Z sont des variables réelles alors que, conceptuellement, elles ne prennent que des valeurs binaires. Néanmoins, définir les variables Z comme variables binaires n'est pas nécessaire car leur comportement est forcé par les contraintes (2.8) et (2.9).

L'objectif du *PTP* est de trouver le positionnement optimal de tous les blocs le long de la séquence. La fonction objectif suivante propose de trouver la meilleure combinaison de variables Y et Z, c'est-à-dire de trouver le chemin augmenté optimal dans le graphe G ou encore d'optimiser la somme des scores associés aux sommets et aux arcs activés. Dans notre cas, nous considérons les scores comme des coûts, c'est pourquoi nous cherchons à minimiser le score total :

$$Min \sum_{k=1}^{m}\sum_{i=1}^{n} C_{ik}Y_{ik} + \sum_{(i,k,j,l)\in E} C_{ikjl}Z_{ikjl} \tag{2.11}$$

Ce modèle est *mixte* puisqu'il utilise des variables Y entières et des variables Z réelles. Une fois défini, un tel modèle peut être résolu par différents logiciels implémentant différentes méthodes. Citons le logiciel CPLEX de la société Ilog[1], le langage AMPL[2], ou encore les bibliothèques libres COIN-OR[3] et GLPK [4].

Ces différents logiciels permettent de résoudre de nombreux types de problèmes d'optimisation mais cette généricité a un coût. En effet, un algorithme dédié et optimisé pour résoudre efficacement un problème particulier est logiquement plus rapide qu'un algorithme générique. La section suivante présente trois techniques qui, combinées, permettent de résoudre efficacement le PTP [6].

2.3 Méthodes de résolution

Nous présentons ici les trois méthodes que nous avons utilisées pour résoudre nos problèmes MIP : La séparation–évaluation (Branch & bound), la descente de sous-gradient et la relaxation lagrangienne [51].

2.3.1 La méthode par séparation–évaluation

La méthode par séparation–évaluation, ou *branch & bound*, est utilisée pour résoudre des problèmes d'optimisation pour lesquels, d'une part, il n'existe pas d'algorithme permettant de les résoudre en temps polynomial, et d'autre part, l'espace de recherche est immense et le parcourir entièrement n'est pas possible. Afin d'éviter l'énumération de toutes les solutions, la méthode consiste à sub-diviser l'espace des solutions (séparation) puis à le parcourir intelligemment (évaluation).

Classiquement, lors de la phase de séparation, l'espace des solution est divisé en deux. Un arbre binaire est ainsi généré. Dans cet arbre, chaque sous-problème généré est soit une solution triviale du problème (une feuille de l'arbre), soit un ensemble de solutions à parcourir. Afin de parcourir efficacement l'arbre, la phase d'évaluation permet d'élaguer certaines branches.

Dans un nœud de l'arbre, nous ne savons pas calculer la solution optimale du sous-problème en un temps polynomial. Par contre, il est possible de calculer deux valeurs approchées : une sur-estimation et une sous-estimation de la valeur optimale. Le calcul

[1] http ://www-01.ibm.com/software/integration/optimization/cplex/
[2] http ://www.ampl.com/
[3] http ://www.coin-or.org/
[4] http ://www.gnu.org/software/glpk/

de ces deux valeurs doit évidemment être rapide afin que la recherche soit efficace. Nous pouvons donc borner la valeur de la solution optimale OPT par une borne supérieure BS et une borne inférieure BI à chaque nœud de l'arbre : $BI \leq OPT \leq BS$.

Comme nous savons que la valeur optimale est comprise entre ces deux valeurs, nous pouvons éviter de parcourir certains sous-problèmes. En effet, si A et B sont deux sous-problèmes et si $BI(A) > BS(B)$ alors ce n'est pas la peine de résoudre A car la solution optimale de B sera forcément meilleure.

L'algorithme parcourt ainsi l'arbre des sous-problèmes, en élaguant les branches inutiles, jusqu'à ce qu'il trouve la solution optimale, c'est-à-dire jusqu'à ce que la meilleure sur-estimation soit égale à la meilleure sous-estimation. Si $BI = BS$ et $BI \leq OPT \leq BS$ alors $BI = BS = OPT$, la solution optimale est trouvée.

Dans le pire des cas, tous les sous-problèmes doivent être visités. Comme cela est trop coûteux, une limite de temps est fixée. Ainsi, bien que la solution optimale ne soit pas déterminée, la méthode par séparation–évaluation fourni des bornes permettant de l'estimer.

2.3.2 La relaxation lagrangienne

La relaxation lagrangienne [30] est souvent intégrée dans une méthode par séparation–évaluation. En effet, l'intérêt de la relaxation lagrangienne est d'obtenir les bornes d'un sous-problème de façon efficace, ceci réduisant d'autant le parcours dans l'arbre des sous-problèmes. Nous présentons ici les concepts généraux de la relaxation lagrangienne. Une description plus détaillée est proposée par Monique Guignard [28].

Le principe de la relaxation lagrangienne consiste à identifier les contraintes du modèle qui rendent le problème difficile à résoudre. Ces contraintes déterminées, elles sont relachées, c'est-à-dire qu'elles sont enlevées du modèle. Néanmoins, elles sont intégrées aux coûts de la fonction objectif du problème modulo des multiplicateurs de Lagrange.

Par exemple, soit le problème P défini par la fonction objectif :

$$\underset{x}{Min}\ f(x) \tag{2.12}$$

où x est un vecteur de variables binaires (ou entières). Cette fonction est soumise aux contraintes suivantes :

$$Ax \leq b \tag{2.13}$$
$$Cx \leq d \tag{2.14}$$
$$x \in X \tag{2.15}$$

où A et C sont des matrices, b et d sont des vecteurs, et $X = \{0, 1\}$. La valeur optimale du problème P est notée $v(P)$. Le problème P peut également s'écrire sous une forme plus compacte :

$$P = \underset{x}{Min} \{f(x) \,|\, Ax \leq b,\, Cx \leq d,\, x \in X\}$$

Supposons que ce problème soit difficile à résoudre mais, qu'en enlevant les contraintes (2.13), il devienne facile à résoudre. Nous avons alors la possibilité d'utiliser la relaxation lagrangienne. Nous allons relâcher les contraintes (2.13) en les intégrant à la fonction de score à l'aide de multiplicateurs de Lagrange. Les contraintes (2.13) sont dites *dualisées*. Soit λ un vecteur de coûts non négatifs. Le problème relaché, noté $LR_\lambda(P)$ est alors défini par la fonction objectif :

$$LR_\lambda(P) = \underset{x}{Min}\{f(x) + \lambda(Ax - b) | Cx \leq d,\, x \in X\} \tag{2.16}$$

La solution optimale de ce problème relâché est facilement calculable mais son score est sur-évalué, c'est-à-dire inférieur à $v(P)$ puisqu'il s'agit d'une minimisation : $v(LR_\lambda(P)) \leq v(P)$. Afin de trouver une solution la plus proche possible de $v(P)$, il faut donc trouver la valeur maximale du problème relaché $LR_\lambda(P)$ en fonction des multiplicateurs de Lagrange λ. Il s'agit d'un autre problème, appelé le dual lagrangien de P et noté LR. Il est défini par la fonction objectif :

$$LR = \underset{\lambda \geq 0}{Max}\; v(LR_\lambda) \tag{2.17}$$

Résoudre le problème LR permettra de trouver la borne inférieure la plus proche possible de $v(P)$. Pour résoudre ce problème, une nouvelle fois, de nombreuses méthodes existent. Nous présentons ici la descente de sous-gradient.

2.3.3 La descente de sous-gradient

La descente de sous-gradient est une technique connue depuis les années 1970 [30]. C'est une méthode itérative qui, à chaque itération k, consiste à modifier le vecteur λ d'un certain *pas* dans la direction du sous-gradient de $v(LR^k_\lambda)$.

Soit x^k la solution optimale du problème relaché LR^k_λ à l'itération k. Le sous-gradient de $v(LR^k_\lambda)$ est défini par $s^k = (Ax^k - b)^T$, $(Ax^k - b)^T$ représentant la transposée du vecteur $(Ax^k - b)$. À partir de ce sous-gradient, nous calculons le pas à effectuer pour passer à l'itération $k + 1$ tel que :

$$\lambda^{k+1} = \lambda^k + \frac{s^k(\eta^* - \eta^k)}{||s^k||^2} \tag{2.18}$$

Dans cette équation, η^k est la valeur de la solution optimale λ^k du problème relaché LR^k_λ et η^* est la valeur de la solution optimale λ^* de LR. Cette définition du "*pas*" garantit que λ^{k+1} est plus proche de λ^* que λ^k à condition que le vecteur λ soit positif

(ou égal à 0).

Cependant, nous ne connaissons pas la valeur de η^* puisque c'est l'optimale que nous recherchons. Nous devons donc utiliser une estimation $\bar{\eta}$ qui sera plus grande ou plus petite que l'optimale. Au bout d'un certain nombre d'itérations, si la valeur de la fonction objective ne bouge pas, nous pouvons supposer que $\bar{\eta}$ est sur-estimée. En conséquence, il faudrait réduire la distance $\bar{\eta} - \eta^k$. Pour ce faire, nous introduisons un facteur $\epsilon_k \in (0, 2)$ dans la formule 2.18 :

$$\lambda^{k+1} = \lambda^k + \frac{s^k \cdot \epsilon_k(\bar{\eta} - \eta^k)}{||s^k||^2} \tag{2.19}$$

Si auncune amélioration n'est observée pendant un certain temps, la valeur de ϵ_k est diminuée pour compenser la sur-estimation de l'optimale.

La convergence d'une descente de sous-gradient n'est pas prévisible. Pour certains problèmes, elle va converger rapidement vers la solution optimale. Dans d'autres cas, elle aura un comportement erratique tant dans la génération des multiplicateurs que dans la valeur du problème relaché. Dans les cas favorables, on observera un comportement en dent de scie dans les premières itérations puis une amélioration monotone et une convergence asymptotique vers, nous l'espérons, la valeur optimale de la relaxation lagrangienne. Dans les mauvais cas, le schéma en dents de scie continue après les premières itérations, voire pire, la valeur du problème relaché se détériore.

Pour résoudre ce problème, nous pouvons essayer d'améliorer le calcul du "*pas*" [12] ou utiliser une autre méthode pour résoudre le problème LR comme la méthode "*Dual ascent*" [29], les "*Cutting Planes*" [14, 36], la génération de colonnes [45, 72], ou les méthodes par faisceaux ("*Bundle*" en anglais) [41, 81]. Une description plus détaillée de ces différentes méthodes est au-delà du sujet de ce manuscrit, c'est pourquoi nous laissons le lecteur intéressé se reporter au chapitre de Monique Guignard [28].

2.4 Ce qu'il faut retenir de ce chapitre

Un problème d'optimisation combinatoire comme le PTP peut se modéliser sous la forme d'un modèle *MIP* consistant en une fonction objectif et des contraintes sur cette fonction. La méthode par séparation-évaluation peut être utilisée pour résoudre un tel problème. Dans cette méthode, l'espace des solutions est découpé en sous-problèmes générant ainsi un arbre binaire. Pour chaque sous-problème, il est nécessaire de borner la solution optimale afin de parcourir intelligemment l'arbre (élagage de certaines branches) et ainsi d'éviter l'énumération de toutes les solutions.

Les bornes d'un sous-problème peuvent être générées en trouvant la solution d'un problème relaché par relaxation lagrangienne. Une relaxation lagrangienne consiste à relacher les contraintes qui rendent le problème difficile à résoudre. Ces contraintes sont

prises en compte dans la fonction objectif par l'ajout d'un coût lié à des multiplicateurs de Lagrange. La solution optimale d'un problème relaché par relaxation lagrangienne peut être recherchée par une descente de sous-gradient.

La combinaison de ces différentes techniques de PLNE ont permis de résoudre efficacement le problème du PTP [4, 6]. Dans la section suivante, nous proposons plusieurs modélisations d'une extension du PTP permettant les délétions dans la structure alignée.

Chapitre 3

FROSTO : un outil pour la reconnaissance de repliements

FROSTO, pour *Fold Recognition Oriented Search Tool (Object oriented version)*, est un logiciel permettant de réaliser des alignements entre protéines. Il est basé sur le logiciel FROST, réalisé en 2002 par Antoine Marin et al. [44]. Dans ce chapitre, après une courte présentation de FROST, nous présentons les raisons qui nous ont conduits à concevoir FROSTO. Nous décrivons ensuite les différentes possibilités offertes par FROSTO ainsi que les améliorations obtenues grâce à la parallélisation des calculs.

3.1 FROST : Fold Recognition Oriented Search Tool

FROST est un logiciel de reconnaissance de repliements qui permet de réaliser des alignements entre une séquence requête et une banque de structures de protéines afin de détecter des similarités.

En interne, FROST est conçu comme un système à deux filtres. La séquence passe d'abord dans le premier filtre où elle est alignée avec toutes les protéines de la banque. Ce filtre est basé sur un score local et un alignement local de type programmation dynamique (section 1.3.2). Les n protéines ayant obtenu les meilleurs scores dans ce filtre sont ensuite passées au second filtre. Ce second filtre aligne la séquence requête avec les n protéines sélectionnées par le premier filtre. Le score du second filtre utilise des positions pairées et nécessite ainsi l'utilisation d'un algorithme d'alignement global de type PTP (section 1.3.3). Les scores obtenus sur ce second filtre permettent de trier à nouveau les protéines et cette liste finale est donnée à l'utilisateur.

L'avantage du fonctionnement en filtres successifs est de pouvoir sélectionner rapidement des candidats puis de raffiner le classement et les alignements avec une fonction de score plus significative. Le premier filtre doit donc être :

- **rapide**, puisqu'il est utilisé sur une grande quantité de données ;

- **sensible**, car un maximum de candidats doit être recruté même si de nombreuses erreurs sont présentes ;
- **pas nécessairement spécifique** car les erreurs seront compensées par le second filtre.

Cependant, si le premier filtre n'est pas assez sensible, c'est-à-dire si il ne recrute pas de "bons" candidats, le deuxième filtre n'apporte pas ou peu d'information. La sensibilité de la méthode dépend donc fortement de la sensibilité du premier filtre.

Afin d'étendre les possibilités de FROST, nous avons souhaité ajouter des méthodes d'alignements et des fonctions de scores différentes. Néanmoins, la structure du programme et les dépendances très fortes entre les différentes parties du code ne permettaient pas d'ajouter simplement des fonctions de scores.

De plus, le "*package*" FROST contenait un grand nombre de logiciels annexes, de scripts, et de fichiers de configuration rendant son installation compliquée et fastidieuse. C'est pourquoi nous avons souhaité rendre FROST modulaire et facilement extensible.

3.2 De FROST à FROSTO

Lors de la conception de FROSTO, nous avons tout d'abord séparé la préparation des données de leur utilisation. FROSTO lit en entrée un certain nombre de fichiers dans des formats prédéfinis. En conséquence, peu importe la façon dont ces fichiers sont créés. FROSTO utilise par exemple des fichiers d'alignements multiples. Ces fichiers peuvent être obtenus par n'importe quel logiciel d'alignements multiples (MUSCLE, DIALIGN, etc) du moment que le format est respecté. L'utilisateur est donc libre d'utiliser le logiciel de son choix pour générer les fichiers nécessaires au fonctionnement de FROSTO.

Nous avons ensuite choisi d'enlever le système des filtres. Désormais, une instance I de FROSTO consiste à aligner une séquence requête S sur une banque de protéines $\{P_1...P_m\}$ en utilisant une fonction de score F_i et un algorithme d'alignement A_j. En sortie, FROSTO produit la liste L de tous les alignements obtenus ordonnés selon leurs scores. Cette simplification nous a permis d'identifier le module élémentaire (F_i, A_j). Une instance de FROSTO se résume donc par ses entrées, le module utilisé et la sortie :

$$I : (S, \{P_1...P_m\}) \rightarrow (F_i, A_j) \rightarrow L$$

Cette simplification possède deux avantages. Tout d'abord, elle permet de reproduire le fonctionnement en filtres de FROST. Il suffit de lancer une instance I_1 de FROSTO avec un premier module, de sélectionner les n premières protéines de la liste produite et de les utiliser dans une instance I_2 avec un autre module.

Le deuxième avantage est que chaque module est indépendant des autres. Par exemple, si nous voulons ajouter une fonction de score, il suffit de créer un module contenant cette fonction et un algorithme d'alignement. Les formats d'entrée et de sortie ne changent pas.

Enfin, Poirriez et al. ont montré que la parallélisation des alignements dans FROST apportait un réel gain de performance au niveau des temps de calcul [55]. Nous avons donc conçu FROSTO en supposant que chaque alignement pourrait s'exécuter en parallèle sur une grille de calcul.

Les sections suivantes sont consacrées à la description des modules de FROSTO (fonctions de scores et méthodes d'alignement). Nous présentons également les gains apportés par la parallélisation du code.

3.3 FROSTO : Fold Recognition Oriented Search Tool Object oriented

FROSTO est un logiciel écrit en C++ que j'ai réalisé en collaboration avec :

- **Guillaume Launay** qui était en contrat post-doctoral sur le projet ANR PROTEUS. Ensemble, nous avons défini le design général de FROSTO, les différents modules et la parallélisation. Il a en particulier implémenté les fonctions de score utilisées dans FROSTO.
- **Yavor Vutov**, qui était ingénieur de recherche sur le projet ANR PROTEUS, a implémenté la partie parallélisation de FROSTO et a également mis en place l'architecture logicielle qui a servi au développement des modules.
- **Alexandre Cornu**, qui était ingénieur expert sur le projet ANR BIOWIC, a participé à la mise au point et aux tests de la partie parallélisation initialement implémentée par Yavor Vutov.
- **Noël Malod-Dognin**, qui était doctorant dans l'équipe Symbiose, a participé à l'élaboration de l'algorithme d'alignement local. Nous avons collaboré sur l'implémentation de la méthode par séparation–évaluation et de la descente de sous-gradient définies dans le chapitre 5.

3.3.1 Parallélisation

Lorsqu'une séquence requête est alignée avec n protéines de la banque de données, les n alignements sont indépendants. Ils sont donc très facilement parallélisables. À partir de la définition d'une instance de FROSTO, nous avons défini une tâche unitaire parallélisable (S, P_k, F_i, A_i) qui correspond à aligner la séquence S avec la protéine P_k en utilisant la fonction de score F_i et l'algorithme d'alignement A_j.

Comme dans tout algorithme parallèle, il faut trouver un bon équilibre entre calcul et communication pour ne pas perdre de temps. Dans cet objectif, les tâches unitaires

sont distribuées équitablement entre différents processus parallèles, appelés clients. La quantité de clients est un paramètre réglable en ligne de commande.

La parallélisation de FROSTO a été programmée avec les librairies *pthread*[1] et *MPI*[2]. La librairie pthread est utilisée pour paralléliser des calculs sur une seule machine alors que la librairie MPI permet de distribuer des calculs sur plusieurs machines. Nous avons testé ces deux librairies sur différents nombres de nœuds (un nœud = une machine contenant 8 cœurs). Nous définissons quatres méthodes : pthread sur 1 nœud (Pthread/1), MPI sur un nœud (MPI/1), MPI sur 4 nœuds (MPI/4) et MPI sur 8 nœuds (MPI/8).

La parallélisation a été testée sur deux jeux de tests en mesurant l'accélération obtenue par rapport à l'algorithme séquentiel. Le premier jeu de tests (J_1) consiste à aligner la structure 1a0iA avec sa propre séquence puis avec 200 séquences de même taille générées aléatoirement. La protéine 1a0iA est de longueur 353 et contient 23 SSEs. Afin de trouver le meilleur équilibre entre communication et calculs, nous avons observé l'accélération obtenue en fonction du nombre de clients et du nombre de nœuds. Les résultats de cette expérience sont présentés dans la figure 3.1.

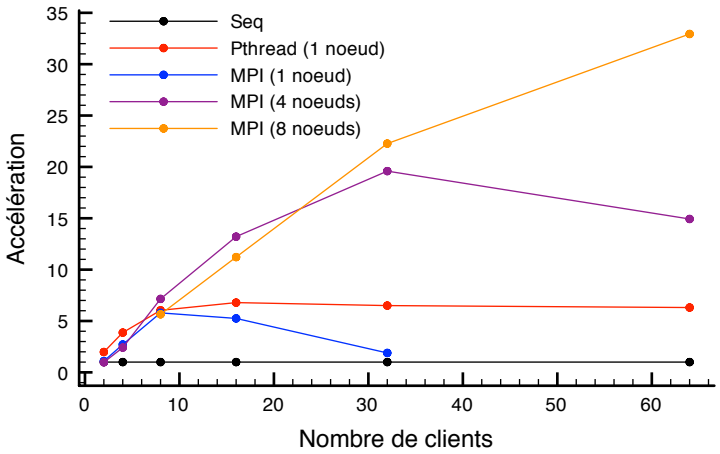

FIG. 3.1 – **Comparaison des différents paramètres de parallélisation sur** J_1. Le jeu de tests J_1 consiste à aligner la structure 1a0iA avec sa propre séquence puis avec 200 séquences de même taille générées aléatoirement. Un point représente une instance (201 alignements). Chaque instance est placée sur le graphique en fonction du nombre de clients alloués (en abscisses) et de l'accélération obtenue par rapport à une exécution séquentielle.

Nous remarquons que l'accélération optimale ($\times 33$) est obtenue pour la parallélisation utilisant la librairie MPI sur 8 nœuds avec 64 clients. Nous remarquons également que

[1] http ://www.opengroup.org/onlinepubs/007908799/xsh/pthread.h.html
[2] http ://www.mcs.anl.gov/research/projects/mpi

sur un seul nœud, si nous augmentons le nombre de clients, la librairie pthread est plus rapide que MPI.

Le deuxième jeu de tests (J_2) consiste à aligner la structure 2volB (15 SSEs) avec la séquence 1h6zA (920 acides aminés) puis avec 200 séquences de la même taille générées aléatoirement. Les résultats de cette expérience sont présentés dans la figure 3.2.

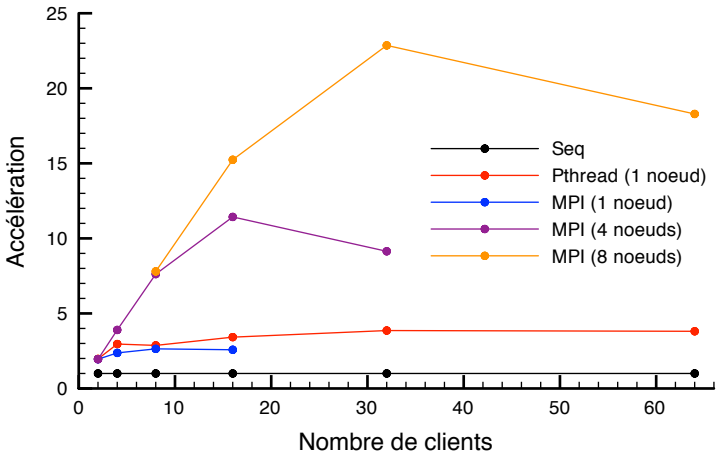

FIG. 3.2 – **Comparaison des différents paramètres de parallélisation sur** J_2. Le jeu de tests J_2 consiste à aligner la structure 1volB avec la propre séquence 1h6zA puis avec 200 séquences de même taille générées aléatoirement. Un point représente une instance (201 alignements). Chaque instance est placée sur le graphique en fonction du nombre de clients alloués, en abscisse, et de l'accélération obtenu par rapport à une exécution séquentielle, en ordonnée.

Nous remarquons que la meilleure accélération ($\times 23$) est obtenue par la parallélisation MPI avec 32 clients sur 8 nœuds. Au-delà de cette valeur, si des clients sont ajoutés, l'accélération est moindre. Cela signifie que, pour des instance difficiles, il n'est pas nécessaire de multiplier les clients, car cela n'améliore pas la vitesse de calcul.

Comme dans l'expérience précédente, sur un seul nœud, la librairie pthread semble être plus rapide que MPI. Cette caractéristique pourrait être exploitée afin d'accélérer les calculs en distribuant des paquets d'alignements sur plusieurs nœuds, via MPI ou simplement par un script, puis en parallélisant localement leurs calculs via pthread. Cette parallélisation n'a pas encore été implémentée au moment de la rédaction de ce document.

3.3.2 Fonctions de scores

3.3.2.1 Profils statistiques

Dans FROST, Marin et al. ont développé deux fonctions de scores appelées 1D et 3D [44]. Ces fonctions ont été directement intégrées dans FROSTO. Le calcul des scores de ces fonctions est basé sur la mesure de l'information donnée par un évènement y sur l'occurence d'un évènement x [24], notée $I(x;y)$ et calculée comme suit :

$$I(x;y) = log\frac{P(x,y)}{P(x)P(y)}$$

Dans cette formule, $P(x,y)$ est la probabilité conjointe d'apparition des évènements x et y, $P(x)$ est la probabilité d'apparition de l'évènement x, et $P(y)$ est la probabilité d'apparition de l'évènement y. À partir de cette formule, Marin et al. définissent tous les scores de FROST comme décrit ci-dessous.

Fonction de score 1D Les scores 1D de FROST consistent à mesurer la similarité entre deux résidus, chacun étant à une position donnée dans une séquence ou une structure. Pour les résidus appartenant à une structure, nous connaissons également leur état structurel. Les états structuraux sont définis par les éléments de structure secondaire (hélices, feuillets et boucles) et par l'accessibilité au solvant (enfoui ou exposé). Dans cette section, les acides aminés provenant de la séquence requête seront notés en minuscule (r) et les acides aminés de la structure seront notés en majuscule (R).

Dans FROST, le score de substitution d'un acide aminé r_j (à la $j^{\text{ème}}$ position dans la séquence) par un acide aminé R_i (à la $i^{\text{ème}}$ position dans la structure) dans l'état structural E_i est calculé ainsi :

$$s(r_j, R_iE_i) = -2log\frac{P(r_j, R_iE_i)}{P(r_j)P(R_iE_i}$$

Dans cette formule, $S(r_j, R_iE_i)$ est le score de substitution de l'acide aminé R_i dans l'état structural E_i par l'acide aminé r_j. $P(r_j, R_iE_i)$ est la probabilité conjointe d'apparition dans un alignement de l'acide aminé r_j et de l'acide aminé R_i dans l'état structural E_i. Enfin, $P(r_j)$ est la probabilité d'apparition de l'acide aminé r_j et $P(R_iE_i)$ est la probabilité d'apparition de l'acide aminé R_i dans l'état structural E_i. Pour chaque état structural, une matrice de substitution 20×20 est précalculée sur une banque de structures.

Lors d'un alignement, nous ne considérons pas seulement un acide aminé à une position donnée mais un profil dérivé d'un alignement multiple. Le score d'alignement du profil $prof_j$ (à la $j^{\text{ème}}$ position dans la séquence) avec le profil $Prof_i$ (à la $i^{\text{ème}}$ position

dans la structure) dans l'état structural E_i est calculé par la formule suivante :

$$S(prof_j, Prof_i E_i) = \sum_{l=1}^{20} F(r_j^l) \sum_{k=1}^{20} F(R_i^k)\, s(r_j^l, R_i^k E_i)$$

Dans cette formule, $S(prof_j, Prof_i E_i)$ est le score correspondant à l'alignement de la position i de la structure avec la position j de la séquence, $F(r_j^l)$ est la fréquence d'apparition du résidu l à la position j dans l'alignement multiple associé à la séquence, et $F(R_i^k)$ est la fréquence d'apparition du résidu k à la position i dans l'alignement multiple associé à la structure.

Fonction de score 3D La fonction de score 3D est similaire à la fonction de score 1D mais au lieu de prendre en compte une seule position, elle s'applique sur des paires de positions. Une paire d'acides aminés aux positions j et j' dans la séquence est notée $r_{jj'}$. Une paire d'acides aminés aux positions i et i' dans la structure et dans les états structuraux respectifs E_i et $E_{i'}$ est notée $R_{ii'}E_{ii'}$. Ainsi, les scores de substitutions sont calculés par la formule suivante :

$$s(r_{jj'}, R_{ii'}E_{ii'}) = -2log\frac{P(r_{jj'}, R_{ii'}E_{ii'})}{P(r_{jj'})P(R_{ii'}E_{ii'})}$$

Dans cette formule, $P(r_{jj'}, R_{ii'}E_{ii'})$ est la probabilité conjointe d'apparition de la paire $r_{jj'}$ et de la paire $R_{ii'}E_{ii'}$, $P(r_{jj'})$ est la probabilité 'apparition de la paire $r_{jj'}$, et $P(R_{ii'}E_{ii'})$ est la probabilité d'apparition de la paire $R_{ii'}E_{ii'}$. Comme pour la fonction de score 1D, ces matrices sont précalculées sur une banque de structures.

Comme pour la fonction de score 1D, lors d'un alignement, les scores de substitutions sont pondérés par les profils des deux protéines alignées. Ainsi, le score d'alignement du profil $prof_{jj'}$ de la paire $r_{jj'}$ avec le profil $Prof_{ii'}$ de la paire $R_{ii'}E_{ii'}$ est donné par la formule suivante :

$$S(prof_{jj'}, Prof_{ii'}E_{ii'}) = \sum_{l=1}^{n_{jj'}} F(r_{jj'}^l) \sum_{k=1}^{n_{ii'}} F(R_{ii'}^k)\, s(r_{jj'}^l, R_{ii'}^k E_{ii'})$$

Dans cette formule, $S(prof_{jj'}, Prof_{ii'}E_{ii'})$ est le score correspondant à l'alignement des positions i et i' de la structure avec les positions j et j' de la séquence, $F(r_{jj'}^l)$ est la fréquence d'apparition de la paire l aux positions j et j' dans l'alignement multiple associé à la séquence, et $F(R_{ii'}^k)$ est la fréquence d'apparition de la paire k aux positions i et i' dans l'alignement multiple associé à la structure.

Le calcul des scores 3D est très long car, dans le pire des cas, toutes les paires d'acides aminés apparaissent dans un alignement multiple. Il y a donc potentiellement 400 paires dans chaque alignement multiple. Si m est le nombre de blocs alignés et n le nombre de

positions possibles pour ces blocs, le nombre de scores 3D à calculer est de complexité $O(16.10^4 \times n^2 \times m^2)$. Il faut noter, en effet, que les scores des paires d'acides aminés A_iA_j et A_jA_i ne sont pas symétriques quand les acides aminés sont dans des environnements différents. Par contre, dans des environnements identiques, les scores des paires A_iA_j et A_jA_i sont identiques. Ces calculs étant indépendants les uns des autres, il est néanmoins possible de les paralléliser afin de les exécuter rapidement.

3.3.2.2 Fonction de score SSE

Dans FROSTO, nous avons ajouté une fonction de score, dite fonction SSE, basée sur les éléments de structure secondaire (SSEs) observés dans la structure et prédits dans la séquence. Cette fonction est basé sur les scores de substitution des éléments de structure secondaire proposé par Wang et Dunbrack [69] et présentés dans le tableau 3.1. Ces scores de substitutions sont notés $s(o_i, p_j)$, ou o_i est le SSE observé à la position i dans la structure et p_j est le SSE prédit à la position j dans la séquence.

	H_o	C_o	S_o
H_p	1.38	-1.86	-3.83
C_p	-1.19	0.81	-0.70
S_p	-3.40	-1.21	1.54

TAB. 3.1 – **Scores de substitution des éléments de la structure secondaire**. Dans ce tableau, les lignes correspondent aux SSEs préditent dans la séquence, et les colonnes correspondent aux SSEs observées dans la structure. La lettre H correspond aux hélices-α, C aux boucles et S aux feuillets-β. Cette matrice a été proposée par Wang et Dunbrack en 2004 [69].

La prédiction de la structure secondaire des séquences est réalisée avant l'utilisation de FROSTO par le logiciel PsiPred [35]. Pour une séquence, PsiPred donne la probabilité $P_j(k)$ d'observer chaque élément de structure secondaire k à chaque position j. Le score d'alignement, d'une position i dans la structure avec une position j dans la séquence, est calculé par la formule suivante :

$$S(i,j) = P_j(H)s(o_i,H) + P_j(C)s(o_i,C) + P_j(S) * s(o_i,S)$$

L'utilisation de PsiPred pour prédire la structure secondaire des séquences n'est pas obligatoire. Pour utiliser la fonction de score SSE, il suffit de produire un fichier dans le format lu par FROSTO et contenant les probabilités d'apparitions des SSEs aux positions de la séquence.

3.3.3 Types d'alignements

Dans FROST, deux types d'alignements étaient proposés. Le premier consistait à aligner deux séquences de protéines grâce à un algorithme classique de programmation

dynamique en utilisant les scores 1D. Il s'agissait donc d'un alignement local séquence–séquence (les trois opérations, substitution, délétion et insertion, étaient possibles).

Le deuxième type d'alignement consistait à aligner une séquence avec une structure de protéine comme dans le problème du PTP. Dans un tel alignement, la longueur de la séquence requête doit égale à $\pm 30\%$ de la taille de la structure. Il s'agit donc d'un alignement global séquence–structure.

Dans FROSTO, nous avons ajouté la possibilité de réaliser des alignements semi-globaux séquence–structure. Ce type d'alignement correspond à un alignement global dans lequel la longueur de la séquence n'est plus limitée à $\pm 30\%$ de la taille de la structure. Le but d'un tel alignement est de détecter des domaines à l'intérieur d'un longue séquence comme présenté en section 1.3.

Enfin, nous avons également ajouté la possibilité de réaliser des alignements locaux séquence–structure. Ce type d'alignement est l'objet de ce travail de thèse et est développé dans les sections suivantes.

3.4 Ce qu'il faut retenir de ce chapitre

FROSTO est un logiciel permettant d'aligner des séquences de protéines avec des structures de protéines. Il est basé sur FROST, un logiciel développé par Marin et al. [44]. FROSTO est une version orientée objet de FROST dans laquelle nous avons modularisé l'utilisation des fonctions de score et des alignements.

Une exécution du programme FROSTO consiste à aligner une séquence requête S sur une banque de protéines $\{P_1 ... P_m\}$ en utilisant une fonction de score F_i et un algorithme d'alignement A_j. Ce fonctionnement se résume par cette formule :

$$I : (S, \{P_1 ... P_m\}) \rightarrow (F_i, A_j) \rightarrow L$$

Dans FROSTO, les alignements sont des tâches indépendantes qui sont donc parallélisables sur un cluster de calcul. Ainsi, FROSTO a été conçu pour pouvoir être utilisé en parallèle. Grâce à cette parallélisation, le calcul de 201 alignements est jusqu'à 33 fois plus rapide qu'en séquentiel.

Dans les alignements proposés par FROSTO, les scores utilisés sont basés sur la mesure de l'information donnée par un évènement y sur l'occurence d'un évènement x [24], notée $I(x;y)$ et calculée comme suit :

$$I(x;y) = log \frac{P(x,y)}{P(x)P(y)}$$

Dans cette formule, $P(x, y)$ est la probabilité conjointe d'apparition des évènements x et y, $P(x)$ est la probabilité d'apparition de l'évènement x, et $P(y)$ est la probabilité d'apparition de l'évènement y. À partir de cette formule, Marin et al. définissent tous les scores 1D et 3D. Les scores 1D mesurent la compatibilité entre une position dans la séquence et une position dans la structure. Les scores 3D mesurent la compatibilité entre une paire de positions dans la séquence et une paire de positions dans la structure.

À ces deux types de scores, nous avons ajouté un score basé sur les éléments de structure secondaire (SSE), observés dans la structure, et prédit dans la séquence. La prédiction de la structure secondaire de la séquence permet d'obtenir pour chaque position dans cette séquence, la probabilité d'être dans une hélice, un feuillet ou une boucle. Ces probabilités permettent de pondérer un score de substitution entre SSEs. Ce score est donné par la matrice de substitution 3×3 proposée par Wang et Dunbrack en 2004 [69].

FROSTO permet de reproduire les alignements proposés dans FROST mais il permet également de réaliser des alignements semi-globaux afin de détecter des domaines structuraux dans de longues séquences, ainsi que des alignements locaux qui sont l'objet de ce travail de thèse.

Chapitre 4

Alignements locaux pour la reconnaissance de repliements

Dans ce chapitre, nous définissons et formalisons les alignements locaux pour la reconnaissance de repliements. La section 4.1 définit tout d'abord la notion d'alignement local utilisant des scores pairés. La section 4.2 détaille les différents modèles que nous avons développés pour réaliser ces alignements locaux. Enfin, la section 4.3 présente une comparaison des différents modèles.

4.1 Formalisation des alignements locaux

En 1994, Lathrop a défini le PTP comme un problème consistant à assigner à chaque bloc d'une carte de contacts généralisée une position relative sur une séquence de protéine (section 1.3.3). Comme présenté dans la figure 4.1, cet assignement oblige à attribuer une position à chaque bloc. Cela signifie qu'il n'y a pas de délétions dans la structure alignée.

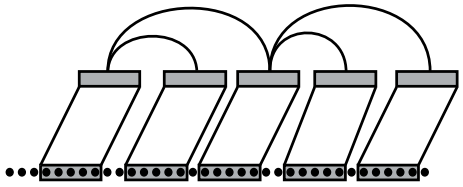

FIG. 4.1 – **Illustration d'un alignement global défini par Lathrop**.

Pourtant, dans le domaine des alignements de séquences, les trois opérations (substitution, insertion et délétion) sont réalisables dans les deux séquences alignées. De plus, dans ce domaine, le type d'alignement le plus utilisé est l'alignement *local* dans lequel, des délétions ne sont pas pénalisées.

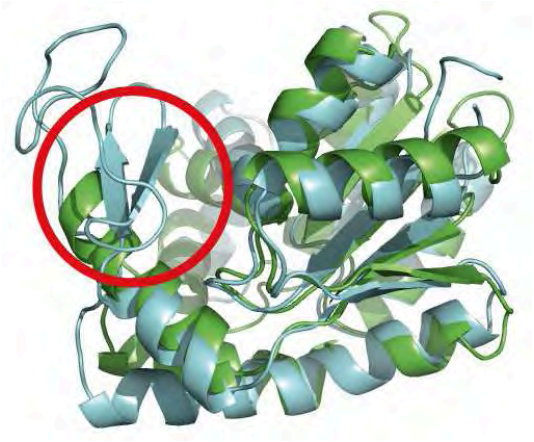

FIG. 4.2 – **Alignement structural entre les protéines 1ukyA et 1dvrA**. L'alignement structural est réalisé par le logiciel VAST [25]. 1ukyA est la protéine colorée en vert, 1dvrA est en bleu. Les éléments de structure secondaire des deux protéines s'alignent deux-à-deux sauf les deux feuillets-β signalés qui ne sont pas présents dans 1ukyA.

L'absence de la délétion dans le PTP entraîne deux restrictions majeures. Tout d'abord, si la séquence est trop courte par rapport à la structure, certains blocs ne pourront pas être positionnés sur la séquence, l'alignement est impossible à réaliser. Ensuite, certaines protéines partagent des structures similaires à un ou deux éléments de structure secondaire près (figure 4.2). Dans le PTP, comme les blocs représentent les hélices-α et les feuillets-β, même si un alignement est réalisable, les blocs surnuméraires risquent de provoquer des décalages dans l'alignement (figure 4.3). Ainsi, la similarité locale entre les deux protéines n'est pas détectée.

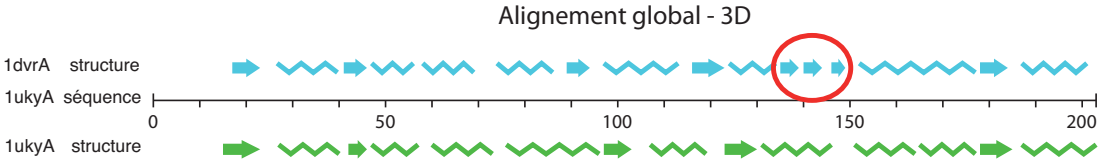

FIG. 4.3 – **Alignement global entre la séquence 1ukyA et la structure 1dvrA**. La séquence de 1ukyA est représentée par un trait noir, la structure de 1dvrA est en bleu. Les hélices-α sont représentées par des zigzags, les feuillets-β par des flèches. La structure de 1ukyA est représentée en vert sous la séquence. Les feuillets-β entourés en rouge provoquent un décalage des éléments de structure secondaire précédents.

Dans ce travail de thèse, nous proposons de relâcher l'obligation d'assigner chaque bloc de la carte de contacts à une position dans la séquence. Autrement dit, un alignement local pour la reconnaissance de repliements consiste à assigner à chaque bloc d'une carte

de contacts généralisée : soit une position sur la séquence requête, soit un *gap* (figure 4.4). Aligner un bloc face à un *gap* correspond à omettre un bloc lors de l'alignement. Dans la suite du manuscrit, nous utiliserons indifféremment les termes d'omission ou de délétion lorsqu'un bloc est aligné face à un *gap*.

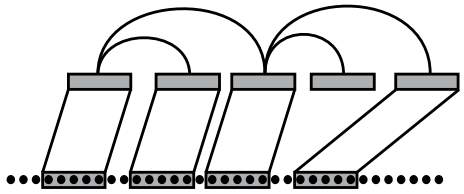

FIG. 4.4 – **Illustration d'un alignement local**.

4.1.1 Graphe d'alignement local (GAL1) : première version

Dans le chapitre 3, le PTP est modélisé sous la forme d'un graphe d'alignement.Dans cette section, nous définissons un graphe d'alignement local (GAL1) sur lequel nous avons développé nos modèles MIP (figure 4.5).

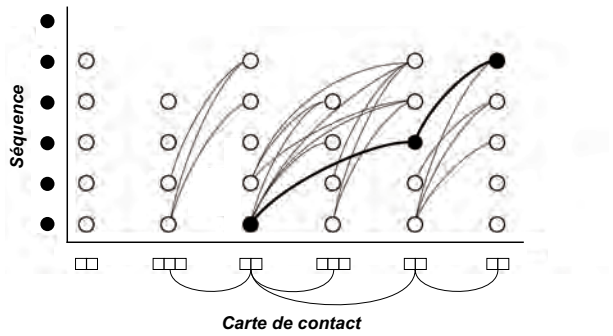

FIG. 4.5 – **Exemple d'alignement dans un graphe d'alignement local GAL1**. En abcisse, une carte de contact représentant une structure de protéine. En ordonnée, une séquence de protéine. Les arcs entre les sommets représentent toutes les interactions entre blocs en fonction de leurs positions. Un exemple d'alignement local est représenté en gras. Les blocs 3,5 et 6 sont activés aux positions 1, 3 et 5. Les blocs 1, 2 et 4 sont omis.

Dans un alignement local, comme certains blocs peuvent être omis, les blocs restants peuvent être positionnés sur toute la longueur de la séquence. Si L_k représente la taille du bloc k, et N la longueur de la séquence, alors chaque bloc peut s'aligner sur $n_k = N - L_k + 1$ positions. Nous ne pouvons donc plus utiliser des positions relatives comme pour le graphe original du PTP. Dans GAL1, les colonnes ont donc des tailles différentes. Activer un sommet ik correspond directement à assigner le bloc k de la structure 3D à la position i sur la séquence. Un bloc k est omis (assigné à un *gap*) si aucun sommet de

la colonne k n'est actif. Trouver un alignement local optimal correspond à activer au plus un sommet dans chaque colonne.

Nous définissons le graphe $GAL1(V_1, E)$ où l'ensemble des sommets $V_1 = \{r_{ik} | 1 \leq k \leq m,\ 1 \leq i \leq n_k\}$ est organisé en colonnes. Chaque colonne k correspond à l'ensemble des positions absolues possibles du bloc k. Une colonne k contient n_k sommets comme décrit précédemment. Ces sommets sont dits "réels", et notés r, par opposition aux sommets factices de GAL2 (voirs section 4.1.2). L'ensemble des arcs $E_1 = \{((i,k),(j,l)) | (k,l) \in I,\ 1 \leq i \leq n_k,\ i + L_k \leq j \leq n_l\}$ représente l'ensemble des interactions entre blocs.

4.1.2 Graphe d'alignement local (GAL2) : deuxième version

Le graphe d'alignement local GAL1 nous a permis de développer le modèle MIP appelé "modèle compact" (voir section 4.2.1). Néanmoins, nous avons eu besoin d'un graphe d'alignement local différent afin de proposer des modèles MIP plus performants. Nous proposons une deuxième version du graphe d'alignement local appelée GAL2. Dans ce graphe, pour chaque colonne k, nous ajoutons $N+1$ sommets représentants la délétion du bloc k. Ces sommets sont dits *factices* car ils représentent une position factice (une délétion). Les sommets factices sont notés d.

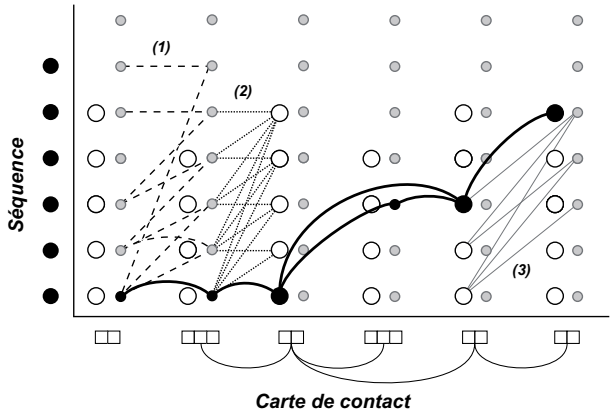

FIG. 4.6 – **Exemple d'alignement dans un graphe d'alignement local GAL2**. En abcisse, une carte de contact représentant une structure de protéine. En ordonnée, une séquence de protéine. Dans ce graphe, les points grisés représentent les sommets factices (d) et les points blancs les sommets réels (r). En plus des arcs de GAL1, trois autres types d'arcs peuvent exister : (1) d'un sommet d à un sommet d, (2) d'un sommet d à un sommet r, (3) d'un sommet r à un sommet d. Comme dans la figure 4.5, les blocs 3,5 et 6 sont activés aux positions 1, 3 et 5. Les blocs 1, 2 et 4 sont omis.

Nous définissons le graphe $GAL2(V_2, E)$ où l'ensemble des sommets $V_2 = V_1 \cup D$ est l'union des sommets de V_1 et de l'ensemble des sommets factices $D = \{d_{ik} | 1 \leq k \leq m,\ 1 \leq i \leq N+1\}$. L'ensemble des arcs E est le même que celui de GAL1. Néanmoins,

dans GAL2, il est possible d'ajouter trois autres types d'arcs en fonction des besoins du modèle. Comme illustré dans la figure 4.6, un arc peut exister : d'un sommet factice à un sommet factice (dd), d'un sommet factice à un sommet réel (dr), d'un sommet réel à un sommet factice (rd).

Les graphes GAL1 et GAL2 sont utilisés dans la section suivante afin de définir les modèles MIP représentant les alignements locaux pour la reconnaissance de repliements.

4.2 Modélisation en problème MIP

Les modèles MIP ont montré leur utilité pour résoudre le PTP [4, 77] surtout lorsqu'ils sont associés à des techniques de résolution efficaces comme l'évaluation–séparation et la relaxation lagrangienne [6]. Notre méthode de travail consiste à développer plusieurs modèles MIP représentant le problème à résoudre. Nous testons ensuite ces modèles grâce au logiciel CPLEX 10.0, un outil permettant de résoudre des modèles MIP de façon générique. Dans cette section, nous présentons cinq modèles MIP représentant les alignements locaux proposés dans la section 4.1.

4.2.1 Modèle compact (CM)

Le modèle compact (CM) est le premier modèle que nous avons développé. Il est basé sur la première version du graphe d'alignement local (GAL1). À chaque sommet ik (ligne i, colonne k), nous associons une variable binaire $Y_{ik} \in \{0, 1\}$. La variable Y_{ik} vaut 1 si le bloc k est aligné à la position i, 0 sinon. De même, nous associons à chaque interaction $ikjl$ (lignes i et j, colonnes k et l) une variable continue Z_{ikjl}. La variable Z_{ikjl} vaut 1, si les blocs k et l sont alignés respectivement aux positions i et j. Comme dans le modèle MYZ, les variables Z ne sont pas binaires car leur comportement va être défini par les contraintes du modèle.

Nous rappelons ici quelques notations utilisées :

- $n_k = N - L_k + 1$ est le nombre de positions du bloc k.
- M est l'ensemble des blocs ($|M| = m$).
- $E = \{((i,k),(j,l))|(k,l) \in I,\ 1 \leq i \leq n_k,\ i + L_k \leq j \leq n_l\}$ est l'ensemble des coordonnées des arcs du graphe d'alignement.

La fonction objectif de ce modèle a la même forme que celle du PTP (2.11) :

$$Min \sum_{k=1}^{m} \sum_{i=1}^{n_k} C_{ik} Y_{ik} + \sum_{(i,k,j,l) \in E} C_{ikjl} Z_{ikjl} \tag{4.1}$$

Cette fonction objectif est soumise aux contraintes suivantes :

$$Y_{ik} \in \{0,1\}, \qquad k \in M,\ i \in [1, n_k] \tag{4.2}$$

$$0 \leq Z_{ikjl} \leq 1, \qquad ((i,k),(j,l)) \in E \tag{4.3}$$

$$\sum_{i=1}^{n_k} Y_{ik} \leq 1, \qquad k \in M \tag{4.4}$$

$$\sum_{j=i+L_k}^{n_l} Z_{ikjl} - Y_{ik} \leq 0, \qquad (k,l) \in I,\ i \in [1, n_k] \tag{4.5}$$

$$\sum_{i=1}^{j-L_k} Z_{ikjl} - Y_{jl} \leq 0, \qquad (k,l) \in I,\ j \in [1, n_l] \tag{4.6}$$

$$Y_{ik} + \sum_{j=1}^{min(n_l, i+L_k-1)} Y_{jl} \leq 1, \qquad 1 \leq k \leq l \leq m,\ i \in [1, n_k] \tag{4.7}$$

$$\sum_{i=1}^{n_k} Y_{ik} + \sum_{i=1}^{n_l} Y_{il} - \sum_{j=L_k+1}^{n_l} \sum_{i=1}^{j-L_k} Z_{ikjl} \leq 1, \qquad (k,l) \in I \tag{4.8}$$

Les contraintes (4.2) et (4.3) définissent les domaines des variables Y et Z. Les contraintes (4.4) correspondent aux constraintes (2.6) de MYZ. Elles imposent à chaque bloc d'être aligné à une seule position, ou omis. Les contraintes (4.7) conservent l'ordre des blocs et empêchent qu'ils se superposent. Les contraintes (4.5) et (4.6) font correspondre l'activation d'un arc avec l'activation des sommets de ses extrémités. Enfin, les contraintes (4.8) imposent l'activation d'un arc entre deux colonnes k et l si un sommet est actif dans chaque colonne. Ces dernières contraintes sont nécessaires car les scores peuvent êtres négatifs ou positifs. S'il n'y a que des scores négatifs, tous les scores des arcs sont favorisant. Les arcs seront forcément activés et les contraintes (4.8) ne sont pas nécessaires. Mais les scores que nous utilisons peuvent être positifs. Dans ce cas, si nous n'ajoutons pas les contraintes (4.8), un arc peut être inactif alors que les sommets à ses extrémités sont actifs.

Ce modèle est le premier que nous ayons développé. Il utilise peu de variables, c'est pourquoi nous l'avons nommé modèle compact. Néanmoins, ce modèle est assez éloigné du modèle MYZ dont il s'inspire. Or, MYZ est un modèle particulièrement efficace pour résoudre le PTP. Nous avons donc souhaité nous rapprocher de ce modèle.

Les différences majeures entre MYZ et CM sont les formes des contraintes. Dans MYZ, les contraintes sont des égalités alors que dans CM, ce sont des inégalités. Nous avons donc développé les modèles suivants en essayant de transformer les inégalités en égalités.

4.2.2 Modèle étendu 1 (EM1)

Dans ce modèle, nous avons transformé les contraintes (4.4) en égalités. Pour cela, nous avons introduit les sommets factices présentés dans GAL2 (section 4.1.2). Les sommets factices sont associés à des variables binaires D_{ik}. Dans ce modèle, omettre le bloc k revient à activer un des sommets factices de la colonne k.

La fonction objectif (4.1) est la même que pour le modèle compact. Celle-ci est soumise aux contraintes (4.2), (4.3), (4.5), (4.6) et (4.8) du modèle CM auxquelles nous ajoutons les contraintes suivantes :

$$D_{ik} \in \{0,1\}, \quad k \in M,\, i \in [1, N+1] \tag{4.9}$$

$$\sum_{i=1}^{n_k} Y_{ik} + \sum_{i=1}^{N+1} D_{ik} = 1, \quad k \in M \tag{4.10}$$

$$\sum_{i=1}^{j} D_{ik} + \sum_{i=1}^{min(j,n_k)} Y_{ik} - \sum_{i=1}^{j} D_{ik-1} - \sum_{i=1}^{j-L_{k-1}} Y_{ik-1} \leq 0, \quad k \in [2,m],\, j \in [1, N+1] \tag{4.11}$$

Les contraintes (4.9) définissent le domaine des variables D. Les contraintes (4.10) ont le même rôle que les contraintes (4.4) du modèle CM. Mais ce sont maintenant des égalités. EM1 est donc plus proche du modèle MYZ.

De plus, les contraintes (4.7) ont disparu du modèle. Elles sont remplacées par les contraintes (4.11). Ces nouvelles contraintes ont exactement le même effet, elles conservent l'ordre des blocs et les empêchent de se chevaucher.

4.2.3 Modèle étendu 2 (EM2)

Dans le modèle EM2, nous avons essayé de transformer les contraintes (4.5) et (4.6) en égalités. Pour cela, nous avons ajouté des arcs entre les sommets réels et les sommets factices. Nous définissons deux nouveaux type de variables binaires :

- $rd_{ikjl} \in \mathbb{R},\ (k,l) \in I,\ (i,j) \in RD$ sont des variables binaires représentant un arc depuis un sommet réel vers un sommet factice ($RD = \{(i,j)\ |(k,l) \in I\ 1 \leq i \leq n_k,\ i + L_k \leq j \leq N+1\}$) ;
- $dr_{ikjl} \in \mathbb{R},\ (k,l) \in I,\ (i,j) \in DR$ sont des variables binaires représentant un arc depuis un sommet factice vers un sommet réel ($DR = \{(i,j)\ |(k,l) \in I,\ 1 \leq i \leq j \leq n_l\}$).

Ce modèle utilise toujours la fonction objectif (4.1) soumise aux contraintes (4.2), (4.3), (4.8), (4.9), (4.10), (4.11) auxquelles nous ajoutons les contraintes :

$$0 \leq rr_{ikjl} \leq 1, \qquad (k,l) \in I,\ (i,j) \in RD \tag{4.12}$$

$$0 \leq dr_{ikjl} \leq 1, \qquad (k,l) \in I,\ (i,j) \in DR \tag{4.13}$$

$$\sum_{j=i+L_k}^{n_l} Z_{ikjl} + \sum_{j=i+L_k}^{N+1} rd_{ikjl} - Y_{ik} = 0, \qquad (k,l) \in I,\ i \in [1, n_k] \tag{4.14}$$

$$\sum_{i=1}^{j-L_k} Z_{ikjl} + \sum_{i=1}^{j} dr_{ikjl} - Y_{jl} = 0, \qquad (k,l) \in I,\ j \in [1, n_l] \tag{4.15}$$

Dans ce modèle, les contraintes (4.12) et (4.13) définissent les domaines des variables rd et dr. Les contraintes 4.5) et (4.6) du modèle compact ont été remplacées par les contraintes 4.14) et (4.15) qui ont le même effet.

4.2.4 Modèle étendu 3 (EM3)

Dans ce modèle, nous avons ajouté des arcs entre les sommets factices :

- $dd_{ikjl} \in \mathbb{R},\ (k,l) \in I,\ (i,j) \in DD$ sont des variables binaires représentant un arc depuis un sommet factice vers un sommet factice ($DD = \{(i,j) \mid 1 \leq i \leq j \leq N+1\}$).

La fonction objectif (4.1) est soumise aux contraintes (4.2), (4.3), (4.9), (4.11), (4.12), (4.13), (4.14) et (4.15). Nous ajoutons les contraintes :

$$0 \leq dd_{ikjl} \leq 1, \qquad (k,l) \in I,\ (i,j) \in DD \tag{4.16}$$

$$\sum_{j=i}^{N+1} dd_{ijkl} + \sum_{j=i}^{n_l} dr_{ijkl} - d_{ik} = 0, \qquad (k,l) \in I,\ i \in [1, N+1] \tag{4.17}$$

$$\sum_{i=1}^{j} dd_{ijkl} + \sum_{i=1}^{j-L_k} rd_{ijkl} - d_{jl} = 0, \qquad (k,l) \in I,\ j \in [1, N+1] \tag{4.18}$$

les contraintes (4.16) définissent le domaine des variables dd. Les contraintes (4.17) et (4.18) remplacent les contraintes (4.8). Ainsi, le modèle EM3 est le plus proche du modèle MYZ.

4.2.5 Modèle étendu 4 (EM4)

Comme le modèle EM3 contient beaucoup de variables et de contraintes, nous avons essayé d'améliorer le modèle EM2 en remplaçant les contraintes (4.8) sans ajouter de variables. Dans ce modèle, la fonction objectif (4.1) est soumise aux contraintes (4.2), (4.3), (4.9), (4.10), (4.11), (4.12), (4.13), (4.14) et (4.15) auxquelles nous ajoutons les

contraintes suivantes :

$$\sum_{j=i}^{n_l} dr_{ijkl} - d_{ik} \leq 0, \qquad (k,l) \in I,\ i \in [1, N+1] \tag{4.19}$$

$$\sum_{i=1}^{j-L_k} rd_{ijkl} - d_{jl} \leq 0, \qquad (k,l) \in I,\ j \in [1, N+1] \tag{4.20}$$

Ces deux types de contraintes permettent d'enlever les contraintes (4.8).

4.3 Comparaison des modèles

4.3.1 Jeu de tests

Les cinq modèles proposés dans la section précédente ont été implémentés dans FROSTO grâce à la bibliothèque de fonctions du logiciel CPLEX 10.0 de la société Ilog. Nous avons créé un jeu de test de 440 alignements (10 séquences requêtes pour 44 structures cibles). Les protéines utilisées dans ce jeu de test sont présentées dans le tableau 4.1.

Séquences requêtes	Structures cibles
1b4aA	1bjaA, 1c20A, 1cf7A, 1ct5A, 1d5vA
1bjaA	1d8jA, 1dp7P, 1e85A, 1ei7A, 1fi2A
1c20A	1fltX, 1g4mA, 1gtfA, 1gvdA, 1gyvA
1cczA	1hc8A, 1hcrA, 1hlvA, 1hw1A, 1im3D
1ct5A	1irzA, 1ix2A, 1jhgA, 1jr8A
1d8jA	1k5nB, 1kmtA, 1l6pA, 1lddA
1dp7P	1n9pA, 1o6sB, 1osyA, 1pp7U
1e85A	1q0eA, 1rowA, 1svbA, 1tbxA
1ei7A	1uadC, 1zrrA, 2bbyA, 2hftA
1jhgA	2hmzA, 2htsA, 2irfG, 3fapB

TAB. 4.1 – **Jeu de test utilisé pour comparer les modèles d'alignement local**. Les protéines sont nommées par leur code PDB.

Les alignements ont été réalisés sur le cluster de calcul de la plateforme bioinformatique GenOuest[1]. Les ordinateurs de ce clusters sont équipés de processeurs AMD Opteron 2,5GHz (bi-coeur) ayant 4Mo de mémoire vive.

[1] http ://www.genouest.org

4.3.2 Distances relatives à l'optimale

La distance relative à l'optimale (RG), ou "*relative gap*" en anglais, est une valeur très utilisée pour mesurer la qualité des modèles MIP. Cette mesure est basée sur la distance entre la valeur de la solution optimale du problème (OPT) et la valeur d'une solution approchée. La solution approchée peut être obtenue par relaxation linéaire, lagrangienne ou par une technique utilisant des heuristiques. Dans notre cas, nous avons utilisé CPLEX afin de générer une solution approchée par relaxation linéaire (LP). La distance relative est alors définie par la formule :

$$RG = \frac{LP - OPT}{OPT}$$

Plus la distance relative est petite, plus la solution relachée est proche de l'optimale. Un modèle est dit "plus fin" qu'un autre si sa distance relative est plus petite.

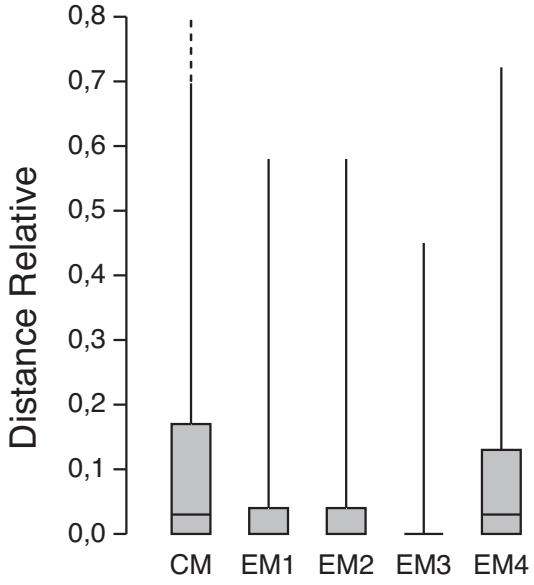

FIG. 4.7 – **Distributions des distances relatives des cinq modèles MIP**. Les distributions sont représentées par des boxplots. Les boîtes grisées représentent 75% des instances car, pour tous les modèles, le minimum est confondu avec le premier quartile. La médiane est également confondue avec le minimum pour tous les modèles étendus (EM). Ceci signifie que, pour 50% des instances, la solution relachée est optimale. Pour des raisons de visibilité, nous avons enlevé le maximum des modèles CM (1,27) et EM4(0,73).

la figure 4.7 présente une comparaison des gaps relatifs des différents modèles. Nous observons que les modèles EM3 et EM4 donnent des gaps relatifs plus fins que les autres modèles. En fait, EM3 est toujours plus fin que n'importe quel autre modèle.Nous remarquons également qu'il donne une distance relative nulle dans 73% des cas. Cela

signifie que dans 73% des cas, la solution optimale est obtenue en résolvant le problème relaché, donc très rapidement. Par comparaison, les taux de distances relatives nulles des autres modèles sont : 42% pour CM, 56% pour EM1 et EM2 et 40% pour EM4.

Nous notons également que EM1 et EM2 donnent exactement les mêmes gaps relatifs. L'ajout de variables dans le modèle EM2 n'a donc pas changé le comportement du modèle.

4.3.3 Temps de calculs

Nous avons ensuite comparé les temps de calculs des différents modèles. Nous remarquons tout d'abord que les modèles EM1 et EM2 sont les plus rapide. Ils ont des temps de calculs très proches. Ceci confirme les observations sur la distance relative : les modèle EM1 et EM2 sont équivalents pour le logiciel CPLEX. En fait, EM1 est toujours plus rapide que n'importe quel autre modèle.

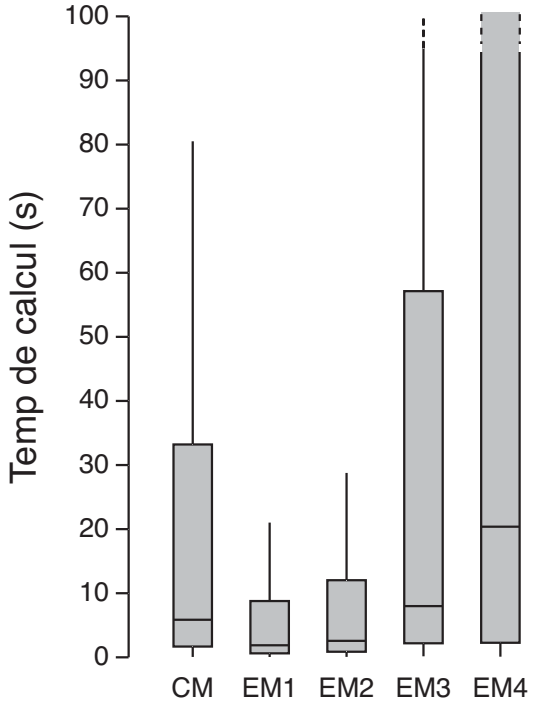

FIG. 4.8 – **Distributions des temps de calcul des cinq modèles MIP**. Les temps de calcul maximum étant très élevés, les lignes supérieures s'arrêtent à la valeur $1,5(q_{75} - q_{25})$. Pour des raisons de lisibilité, nous avons représenté les valeurs jusqu'à 100 secondes. Les temps maximums de chaque modèle sont de 600 secondes.

Nous observons également que le modèle EM3, qui est le plus performant au niveau de la distance relative, est ici moins performant au niveau des temps de calculs. Cela

peut s'expliquer par la quantité de variables et de contraintes du modèle. Enfin, les contraintes exprimées dans le modèle EM4 rendent le problème très long à résoudre.

4.3.4 Nombre de variables et de contraintes

Afin de comprendre plus précisément le comportement des différents modèles, nous avons observé leur nombre de variables (figure 4.9) et de contraintes (figure 4.10).

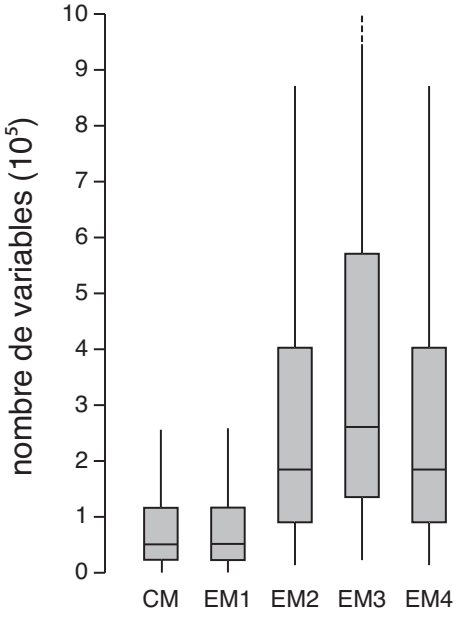

FIG. 4.9 – **Distributions du nombre de variables des cinq modèles MIP**. Les distributions sont représentées par des boxplots. Les boîtes grisées représentent 50% des instances. Les maximums étant très élevés, les lignes supérieures s'arrêtent à la valeur $1,5(q_{75} - q_{25})$. Pour des raisons de lisibilité, le nombre de variables est limité à 10^6. Le nombre maximal de variables pour chaque modèle est : $n(CM) = 8,3.10^5$, $n(EM1) = 8,4.10^5$, $n(EM2) = 26,8.10^5$, $n(EM3) = 36,9.10^5$, $n(EM4) = 26,8.10^5$.

Au niveau du nombre de variables, la première observation évidente est que le modèle CM contient le moins de variables puisqu'il n'y a que des arcs et des sommets réels. Dans le modèle EM1, nous avons doublé le nombre de sommets en ajoutant les sommets factices. En fait, la différence entre CM et EM1 est invisible dans la figure 4.9 car le nombre de sommets est très petit comparé au nombre d'arcs. Dans EM2 et EM4 nous avons ajouté des arcs entre sommets réels et factices. La différence avec EM1 et CM est nettement visible. Enfin, EM3 est le modèle contenant le plus de variables puisqu'il contient tous les sommets et arcs possibles (réels ou factices). En résumé, si $n(i)$ est le nombre de variables du modèle i, alors :

$$n(CM) < n(EM1) < n(EM2) = n(EM4) < n(EM3)$$

La comparaison du nombre de contraintes est présentée dans la figure 4.10. Nous observons que le modèle EM1 contient le moins de contraintes. En effet, l'ajout des sommets factices dans EM1 a permis d'enlever les contraintes (4.7) du modèle CM et de les remplacer par les contraintes (4.11) qui ont le même effet. Le nombre de contraintes (4.7) est proportionnel à m^2N^2 alors que le nombre de contraintes (4.11) est proportionnel à mN (m est le nombre de blocs dans la structure, N est la longueur de la séquence). Il y a donc moins de contraintes dans EM1 que dans CM.

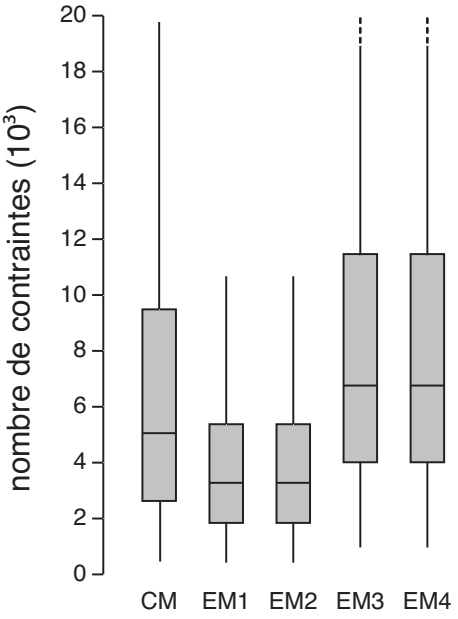

FIG. 4.10 – **Distributions du nombre de contraintes des cinq modèles MIP**. Pour des raisons de lisibilité, le nombre de contraintes est limité à 20.10^3. Le nombre maximal de contraintes pour chaque modèle est : $n(CM) = 99, 7.10^3$, $n(EM1) = 29, 2.10^3$, $n(EM2) = 29, 2.10^3$, $n(EM3) = 57.10^3$, et $n(EM4) = 57.10^3$.

Nous observons également que EM1 et EM2 possédent le même nombre de contraintes, ce qui est cohérent avec la définition des modèles. Enfin, dans EM3 et EM4, le nombre de contraintes devient très grand car nous ajoutons les contraintes (4.17) et (4.18) au modèle EM3, et les contraintes (4.19) et (4.20) dans EM4. Cela revient à doubler le nombre de contraintes dans ces deux modèles.

4.3.5 Ce qu'il faut retenir de ce chapitre

Le problème du PTP consiste à aligner tous les blocs d'une structure de protéine sur une séquence de protéine. Dans ce problème, les blocs sont obligatoirement alignés. Nous proposons de relâcher cette contrainte et de laisser les blocs s'aligner ou non sur une séquence. Un tel alignement est appelé **alignement local** dans le sens où il reproduit

le comportement d'un alignement local de séquences en proposant les trois types de transformations (la substitution, l'insertion et la délétion) et en ne pénalisant pas les délétions.

Nous proposons cinq modèles MIP représentant les alignements locaux. Ces cinq modèles ont été conçus de façon incrémentale à l'aide du logiciel CPLEX. Nous proposons tout d'abord un modèle simple et compact (CM) que nous avons ensuite raffiné en ajoutant des sommets factices (EM1) et des arcs factices (EM2, EM3, EM4). Ces modèles diffèrent par le nombre de variables et de contraintes qu'ils possèdent (figures 4.9 et 4.10).

Sur un jeu de 440 alignements, les expériences montrent que le modèle EM1 est le plus rapide. Néanmoins, nous observons que le modèle EM3 présente une distance relative à l'optimale très petite comparée aux autres modèles. Afin de développer un algorithme dédié pour résoudre un problème, nous prenons traditionnellement le modèle ayant la plus petite distance relative à l'optimale. Mais dans notre cas, le modèle EM3 est aussi très lent, c'est pourquoi nous avons choisi le modèle EM1 pour développer l'algorithme présenté dans la section suivante.

Chapitre 5

Un algorithme dédié pour résoudre les alignements locaux

Les alignements locaux proposés dans le chapitre précédent ont été calculés avec le logiciel CPLEX. Néanmoins, certaines instances sont difficiles à résoudre et CPLEX n'arrive pas à trouver une solution (approchée ou exacte) en un temps raisonnable. Or, pour utiliser les alignements locaux à grande échelle, c'est-à-dire sur un grand jeu de données et sur des instances difficiles, nous devons obtenir des résultats rapidement.

Afin de résoudre rapidement les modèles *MIP* présentés dans la section précédente, nous utilisons un algorithme par séparation–évaluation dans lequel les solutions approchées sont calculées sur une relaxation lagrangienne du modèle EM1.

Dans ce chapitre, nous présentons tout d'abord la stratégie de séparation–évaluation que nous avons développée puis la relaxation lagrangienne du modèle EM1. Nous décrivons ensuite l'algorithme de programmation dynamique utilisé pour résoudre le problème relaché ainsi que les paramètres de la descente de sous-gradient permettant de résoudre son dual. Enfin, nous comparons les résultats de notre algorithme avec CPLEX en termes de temps d'exécution et de distance relative à l'optimale.

5.1 Séparation et évaluation

Dans notre méthode par séparation–évaluation, l'évaluation est réalisée sur la relaxation lagrangienne du modèle EM1. Cette évaluation étant décrite dans les sections suivantes, nous nous intéressons ici à la partie séparation.

Lors de la séparation de l'espace des solutions, une stratégie intéressante est de découper le problème en deux sous-problèmes indépendants et de tailles similaires. Nous avons identifié trois méthodes permettant un tel découpage. Pour une meilleure lisibilité, les exemples de découpage sont illustrés sur des grilles complète plutôt que sur les graphes GAL1 ou GAL2, sauf lorsque cela est nécessaire.

5.1.1 Découpage sur les sommets

Une première méthode simple pour découper l'espace des solutions est de choisir un sommet x et de définir deux sous-problèmes A et B : dans A, x est actif, dans B, x est inactif. Ainsi l'espace des solutions est séparé en deux sous-problèmes indépendants, comme illustré dans la figure 5.1.

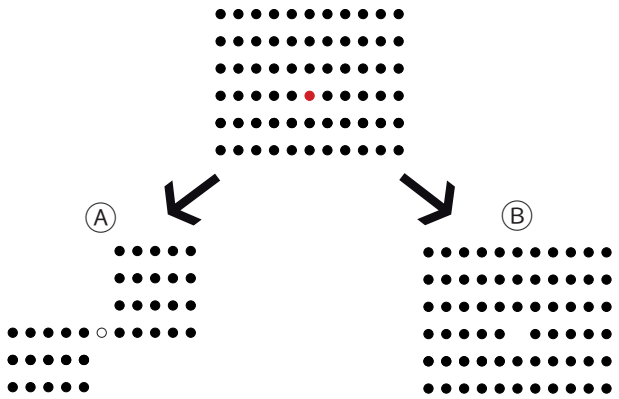

FIG. 5.1 – **Illustration du découpage sur un sommet**. Le problème principal (grille 6×11) est découpé à partir du sommet x (en rouge). Dans le sous-problème **A**, le sommet x est activé, les autres sommets de la colonne correspondante sont donc inactifs. De plus, les contraintes d'ordre et de non-chevauchement des blocs implique que certains sommets ne seront jamais actifs, nous pouvons donc les enlever. Dans le sous-problème **B**, le sommet x n'est pas activé, il est donc enlevé du problème.

Ce type de découpage montre plusieurs limites :

- les deux sous-problèmes ne sont pas de tailles similaires ;
- pour chaque sous-problème, une structure de données doit conserver les choix fait précédemment. La taille de cette structure de données est proportionnelle au nombre de sommets dans le modèle ;
- le nombre de sous-problèmes générés est de complexité $O(2^v)$, v étant le nombre de sommets dans le modèle.

5.1.2 Découpage sur les colonnes

L'espace des solutions peut également être divisé à partir d'une colonne k. Soit A et B deux sous-problèmes. Dans A, la colonne k est active, un sommet de cette colonne sera donc obligatoirement activé. Dans B, la colonne k est inactive, aucun sommet ne sera activé. Le découpage basé sur les colonnes est illustré dans la figure 5.2.

Le choix de la colonne k est basé sur le nombre d'interactions. En effet, plus une colonne possède d'interactions, plus elle participe au score. En conséquence, choisir d'activer, ou non, une telle colonne peut entraîner de fortes variations du score global.

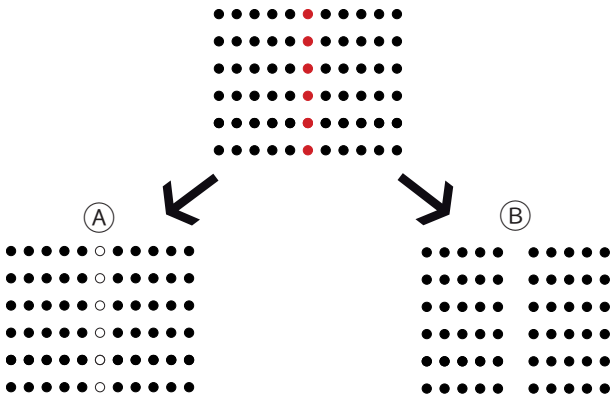

FIG. 5.2 – **Illustration du découpage sur une colonne**. Le problème principal est découpé à partir de la colonne k choisie (en rouge). Dans le sous-problème **A**, la colonne k est activée, un de ses sommets doit être actif. Dans le sous-problème **B**, la colonne k n'est pas activée, elle est donc enlevée du problème.

Nous espérons ainsi pouvoir couper des branches proches de la racine de l'arbre de recherche.

Le découpage sur les colonnes permet d'utiliser moins de ressources que le précédent découpage. En effet, pour chaque sous-problème, il suffit de conserver les choix effectués précédemment dans un vecteur de taille m, où m est le nombre de colonnes. De plus, lorsqu'une colonne est active, cela impose qu'un sommet soit activé dans cette colonne et cela implique un découpage du graphe.

Si un sommet est obligatoirement activé dans la colonne k, alors les sommets des colonnes suivantes ne peuvent s'activer qu'à partir de la ligne L_k. De même, les sommets des colonnes l précédentes ne peuvent s'activer qu'en dessous de la ligne $n_l - L_k$. Le découpage induit par une colonne active réduit ainsi l'espace des solutions (figure 5.3).

Un défaut de ce découpage est qu'il ne permet pas d'obtenir des sous-problèmes triviaux, c'est-à-dire correspondant à une solution unique. En effet, lorsqu'il ne reste que des colonnes actives dans le graphe, nous ne savons pas quels sont les sommets à activer. Il faut encore découper le problème pour d'obtenir une solution triviale. Nous montrons que ce défaut peut être compensé par le découpage présenté ci-dessous.

5.1.3 Découpage sur le graphe

La dernière méthode consiste à découper le graphe de façon à créer deux sous-graphes contenant un nombre de sommets similaires. Pour réaliser ce découpage, nous cherchons un sommet qui servira de pivot. Le graphe est ensuite découpé au niveau de ce pivot comme décrit dans la figure 5.4. Ainsi, les deux sous-problèmes sont de tailles similaires.

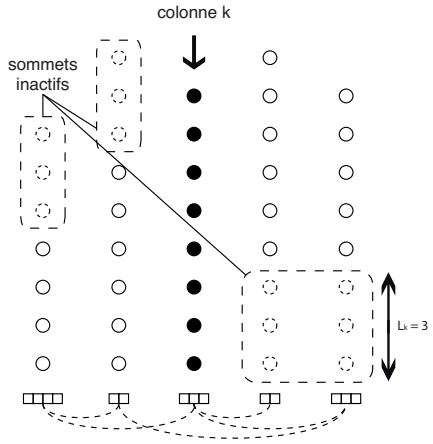

FIG. 5.3 – **Illustration du découpage induit par l'activation d'une colonne sur un graphe GAL1**. Les contraintes d'ordre et de non-chevauchement impliquent l'inactivation des sommets en pointillés si la colonne centrale est activée.

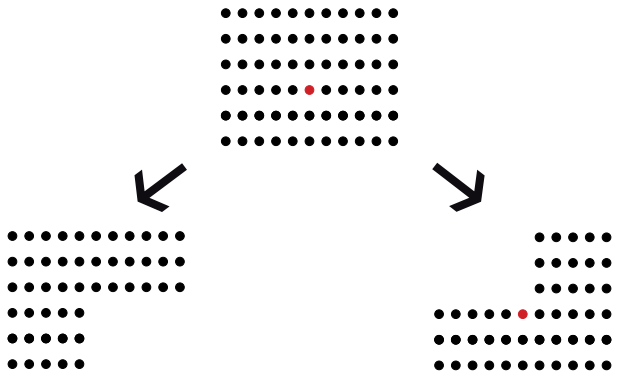

FIG. 5.4 – **Illustration du découpage d'un graphe**. Le graphe entier est à la racine de l'arbre. Il est ensuite découpé en deux sous-graphes par un algorithme qui choisi le sommet pivot le plus équilibré (en rouge), c'est-à-dire pour lequel les deux sous-graphes ont un nombre de sommets similaire.

Dans un problème d'alignement global, comme chaque colonne contient un sommet actif, ce découpage permet d'obtenir des sous-problèmes indépendants. Mais cette propriété n'est plus vérifiée dans le cas des alignements locaux. En effet, comme illustré dans la figure 5.5, deux sous-problèmes peuvent partager des solutions identiques car certaines colonnes sont inactives. Ce découpage n'est donc pas directement applicable à notre problème.

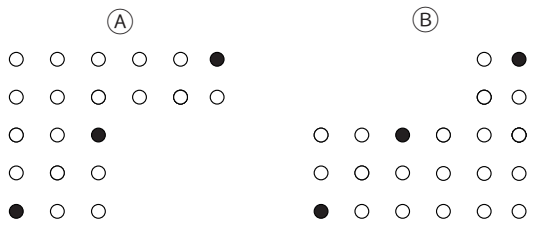

FIG. 5.5 – **Exemple de deux solutions identiques dans deux sous-problèmes différents**. Les colonnes 2,4 et 5 étant omises, la solution (sommets noirs) est identique dans les deux sous problèmes.

Néanmoins, dans le découpage sur les colonnes présenté dans la section précédente, lorsqu'un problème n'est plus divisible, il ne reste que des colonnes actives. Dans ce cas, un découpage sur le graphe est fonctionnel. Dans notre algorithme, nous commençons donc par découper le problème à l'aide d'un découpage sur les colonnes puis, lorsque ce n'est plus possible, nous découpons les sous-problèmes obtenus par un découpage sur le graphe.

5.2 Relaxation Lagrangienne du modèle EM1

Afin d'utiliser une relaxation Lagrangienne sur le modèle EM1, la première étape consiste à trouver les contraintes qui le rendent difficile à résoudre. Comme pour le PTP, la difficulté vient principalement des arcs entre les blocs [39]. En effet, sans ces arcs, le PTP peut se résoudre par une programmation dynamique [6].

Plus particulièrement, les contraintes rendant le problème difficile sont celles qui font correspondre les extrémités des arcs avec les sommets activés. Dans le modèle EM1, ce sont les contraintes (4.6), (4.5) et (4.8). Si nous relâchons ces contraintes, le problème peut être résolu en un temps polynomial. Cependant, il n'est pas nécessaire de relâcher tous les types de contraintes. Par exemple, si nous ne relâchons que les contraintes (4.6) et (4.8), cela revient à libérer l'extrémité droite des arcs. Dans ce cas, il n'est pas nécessaire de faire correspondre l'activation d'un sommet avec l'activation des arcs entrant dans ce sommet. Le problème peut être résolu en un temps polynomial par un algorithme de programmation dynamique.

Nous avons donc choisi de relacher les contraintes (4.6) et (4.8) du modèle EM1. Les contraintes (4.6) s'appliquent sur les sommets ayant des arcs entrant. Ainsi, un multiplicateur lagrangien $\lambda_{kjl} \geq 0$ est associé à chaque triplet (k, j, l) tel que $(k, l) \in I$ et $L_k < j < n_l$. L'ensemble des multiplicateurs lagrangiens associés aux contraintes (4.6) est défini comme suit :

$$\Lambda = \{\lambda_{kjl} | (k, l) \in I,\, L_k < j < n_l\}$$

Ainsi, les contraintes (4.6) sont intégrées à la fonction objectif sous la forme suivante :

$$\sum_{\lambda_{kjl} \in \Lambda} \lambda_{kjl} \left(\sum_{i=1}^{j-L_k} Z_{ikjl} - Y_{jl} \right)$$

Les contraintes (4.8) s'appliquent sur les paires de colonnes. À chaque paire $(k, l) \in I$ est associée un multiplicateur lagrangien $\gamma_{kl} \geq 0$. L'ensemble des multiplicateurs lagrangiens associés aux contraintes (4.8) est défini comme suit :

$$\Gamma = \{\gamma_{kl} | (k, l) \in I\}$$

Ainsi, les contraintes (4.8) sont intégrées à la fonction objectif sous la forme suivante :

$$\sum_{\gamma_{kl} \in \Gamma} \gamma_{kl} \left(\sum_{i=1}^{n_k} Y_{ik} + \sum_{i=1}^{n_l} Y_{il} - \sum_{j=L_k+1}^{n_l} \sum_{i=1}^{j-L_k} Z_{ikjl} - 1 \right)$$

Afin de simplifier l'écriture le fonction objectif, les multiplicateurs lagrangiens λ et γ sont directement ajoutés aux scores C de la fonction objectif. Ainsi, dans le problème relaché, le score associé à une variable Y_{ik}, noté $C_{ik}^{\lambda\gamma}$, est défini par :

$$C_{ik}^{\lambda\gamma} = C_{ik} + \sum_{(l,k) \in I} (\gamma_{lk} - \lambda_{lik}) + \sum_{(k,l) \in I} \gamma_{kl}$$

De la même façon, le score associé aux arcs, donc aux variables Z_{ikjl}, est noté $C_{ikjl}^{\lambda\gamma}$ et est défini par :

$$C_{ikjl}^{\lambda\gamma} = C_{ikjl} + \lambda_{kjl} - \gamma_{kl}$$

Nous définissons ainsi le problème relaché ER1 par la fonction objectif suivante :

$$Min \sum_{k=1}^{m} \sum_{i=1}^{n_k} C_{ik}^{\lambda\gamma} Y_{ik} + \sum_{(i,k,j,l) \in E} C_{ikjl}^{\lambda\gamma} Z_{ikjl} - \sum_{(k,l) \in I} \gamma_{kl} \tag{5.1}$$

Cette fonction objectif est soumise aux contraintes (4.2), (4.3), (4.5), (4.9), (4.10), et (4.11). Le problème ER1 ainsi défini peut être résolu par un algorithme de programmation dynamique comme décrit dans la section suivante.

5.3 Résolution du problème relaché

Dans le problème ER1, les contraintes (4.6) et (4.8) n'étant plus présentes, l'activation d'un arc sortants du sommet (i,k) vers la colonne l n'est plus contraint par l'activation des sommets dans la colonne l. Ainsi, le score d'un sommet est donné par la formule

$$s(i,k) = C_{ik}^{\lambda\gamma} + GetArcOut(i,k)$$

où $C_{ik}^{\lambda\gamma}$ est le score associé au sommet (i,k) et $GetArcOut(i,k)$ est une procédure donnant la somme des scores des arcs activés sortants du sommet (i,k). Le détail de cette procédure est présentée en section 5.3.2.

L'algorithme par programmation dynamique que nous avons utilisé pour résoudre le problème ER1 est présenté dans la section suivante.

5.3.1 Programmation dynamique globale

5.3.1.1 Forme de la matrice

Dans une matrice de programmation dynamique, appelée matrice DP, une case (i,k) contient le score du chemin optimal depuis la case $(1,1)$ jusqu'à la case (i,k). Si cette matrice représente un alignement entre des séquences A et B, le chemin optimal de $(1,1)$ à (i,k) représente l'alignement optimal entre les i premiers éléments de la séquence A et les k premiers éléments de la séquence B. Cela signifie donc que les i (resp. k) premiers éléments de la séquence A (resp. B) sont utilisés.

Dans les graphes d'alignement GAL1 et GAL2 (section 4.1), la position d'un bloc sur la séquence est définie par la position de son premier acide aminé (section 1.3.3). Par conséquent, si un bloc k est en position i alors les $i + L_k$ premiers acides aminés de la séquence sont utilisés (figure 5.6.A). Ceci est incompatible avec la définition de la matrice de programmation dynamique précédente.

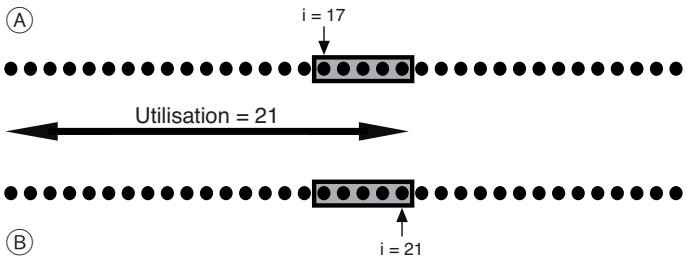

FIG. 5.6 – **Influence de la définition de la position d'un bloc**. A) La position d'un bloc est définie par la position de son premier acide aminé. Cette position ne correspond pas au nombre d'acides aminés alignés dans la séquence. B) La position d'un bloc est définie par la position de son dernier acide aminé. Cette position correspond au nombre d'acides aminés alignés dans la séquence.

Afin d'adapter GAL1 et GAL2 dans une matrice de programmation dynamique correcte, la position d'un bloc sur la séquence est définie par la position de son dernier acide aminé. Ainsi, un bloc k positionné en i correspond bien à l'utilisation des i premiers acides aminés de la séquence (figure 5.6.B). Dans ce graphe modifié, un bloc k ne peut être positionné en dessous de la ligne L_k. Le graphe d'alignement modifié est illustré dans la figure 5.7.

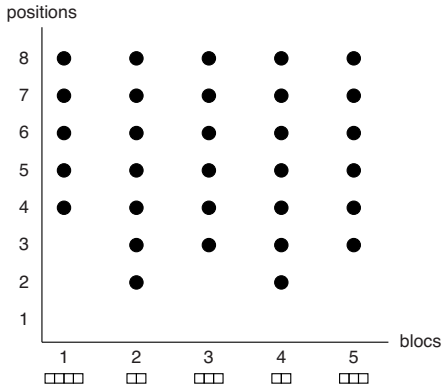

FIG. 5.7 – **Graphe d'alignement modifié pour la programmation dynamique**. La position d'un bloc est définie par son dernier acide aminé. Ainsi, une colonne k est décalée de $L_k - 1$ positions vers le haut par rapport à GAL1.

La matrice DP contient donc $m+1$ colonnes et $N+1$ lignes, la première ligne et la première colonne servant à initialiser la programmation dynamique. Résoudre le problème ER1 dans cette matrice est un problème de complexité $O(mN)$. Une superposition de la matrice DP et du graphe modifié est illustrée sur la figure 5.8.

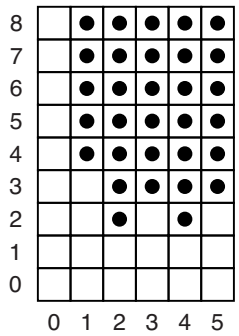

FIG. 5.8 – **Superposition de la matrice de programmation dynamique DP et du graphe d'alignement GAL1 modifié**. Le graphe d'alignement GAL1 modifié est représenté par les points noirs. La ligne et la colonne 0 servent à initialiser la matrice.

5.3.1.2 Mouvements dans la matrice

Dans une matrice de programmation dynamique classique, le score d'une case (i, j) est le meilleur score parmi les suivants :

- le score de la case à gauche $(i, k-1)$ (mouvement **horizontal**) ;
- le score de la case en-dessous $(i-1, k)$ (mouvement **vertical**) ;
- le score de la case en diagonale $(i-1, k-1)$ + le score du sommet $s(i, k)$ (mouvement **diagonal**).

Cette définition s'applique lorsque les éléments alignés sont de même taille, mais dans notre cas, les blocs ont des longueurs différentes. Ainsi, comme les blocs ne peuvent pas se chevaucher et doivent rester dans l'ordre, le mouvement diagonal est décalé. En effet, si un bloc k est aligné en position i, alors le bloc suivant $k+1$ ne peut pas être placé à une position inférieure à $i + L_{k+1}$ sinon, les blocs se chevauchent ou changent d'ordre (figure 5.9).

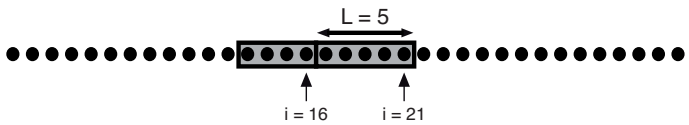

FIG. 5.9 – **Les positions de deux blocs consécutifs sont, au minimum, séparées par la longueur du deuxième bloc**. La longueur du deuxième bloc étant de 5, La position de celui-ci est au minimum de $16 + 5 = 21$.

Dans la matrice DP, le mouvement diagonal vers la case (i, k) est donc calculé à partir de la case $(i - L_k, k-1)$. Cette définition implique que le score d'une case (i, k), où $i < L_k$, ne dépend que des case $(i-1, k)$ et $(i, k-1)$; le mouvement diagonal n'est pas possible.

5.3.2 Calcul des arcs sortants

Dans la section précédente, nous avons défini la procédure $GetArcOut(i, k)$ qui renvoie la somme des scores des arcs activés entre le sommet (i, k) et les colonnes l telles que $(k, l) \in I$. Dans cette procédure, nous voulons activer les arcs ayant le meilleur score sans tenir compte de l'activation des sommets à droite des arcs. Pour cela, nous proposons les deux méthodes décritent ci-dessous.

5.3.2.1 Programmation dynamique locale

Pour définir les meilleurs arcs sortants d'un sommet (i, k), un algorithme de programmation dynamique peut être utilisé. À partir du sommet (i, k), nous définissons une matrice T_{ik} (figure 5.10) dont le nombre de lignes est $n_l = N + 1 - i$ et le nombre de colonnes m_l est le nombre d'interactions $(k, l) \in I$. Chercher les meilleurs arcs sortants du sommet (i, k) revient à chercher le meilleur chemin de la case $(0, 0)$ à la case (n_l, m_l)

dans la matrice T_{ik}, en tenant compte des scores des arcs rencontrés. La complexité de cet algorithme est $O(m_l n_l)$.

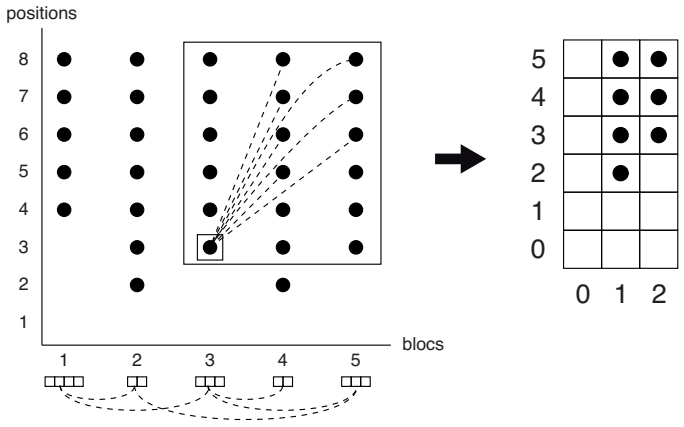

FIG. 5.10 – **Exemple de matrice de programmation dynamique locale**. La matrice de droite est utilisée pour trouver les meilleurs arcs sortants du sommet $(3, 3)$. Si une case contient un point, il y a un arc depuis le sommet $(3, 3)$ jusqu'à cette case et son score est pris en compte lors d'un mouvement diagonal.

Comme dans la programmation dynamique globale, le score de la case (j, l), est le meilleur score parmi :

- le score de la case $(j-1, l)$;
- le score de la case $(j, l-1)$;
- le score de la case $(j - L_l, l-1)$ + le score de l'arc $(i, k)(j, l)$.

Cette définition de la procédure $GetArcOut(i, k)$ garantit que les extrémités droites des arcs activés respectent les contraintes d'ordre (4.11). Néanmoins, cette procédure est appelé environ $m \times N$ fois lors de la programmation dynamique globale. Cela entraîne une complexité globale de l'algorithme proportionnelle à $m \times N \times m_l \times n_l$. m et m_l sont proportionnels au nombre de SSEs dans une structure, N et n_l sont proportionnels à la longueur d'une séquence. La complexité est donc quadratique par rapport à la taille des protéines alignées, ce qui peut entraîner des temps de calculs très longs.

Cette méthode de calcul des arcs sortants est notée DP (*Dynamic Programming*) dans la suite du document.

5.3.2.2 Tas binaires

Dans la programmation dynamique locale décrite précédemment, les contraintes d'ordre sur les extrémités droites des arcs sont respectées. Or le modèle ER1 n'oblige pas à respecter ces contraintes puisque les extrémités droites des arcs ne sont plus reliées aux

sommets. Ainsi, au lieu de choisir les arcs en respectant ces contraintes, nous proposons de choisir les arcs **uniquement** en fonction de leur score.

Afin de choisir les arcs en fonction de leur score, il est nécessaire de les trier. De plus, ce tri doit être efficace car les arcs représentent une grande quantité de données. Pour cela, nous proposons d'utiliser la structure de données appelée *tas binaire* ou *binary heap* en anglais.

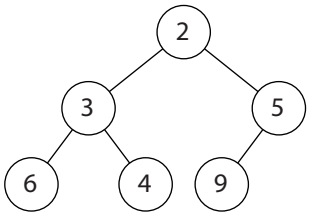

FIG. 5.11 – **Exemple de tas binaire**. Dans ce tas binaire, chaque *père* a une valeur inférieure à ses fils. La racine possède donc la valeur minimale. Il n'y a pas d'ordre entre les fils.

Un tas binaire est un arbre binaire ayant une relation d'ordre total, notée $\prec$, sur la valeur des nœuds, notée v. Dans un tas binaire, la propriété suivante est toujours vérifiée : Si le nœud A est le père du nœud B alors $v(A) \prec v(B)$. Dans notre cas, comme nous cherchons un minimum, la relation d'ordre total "$\prec$" est une relation d'infériorité "$\leq$". Ainsi, nous garantissons que la valeur d'un père est toujours plus petite que celles de ses fils. La racine de l'arbre possède donc la valeur la plus petite et elle est accessible directement. La figure 5.11 présente un exemple de tas binaire.

Un tas binaire possède aussi des caractéristiques intéressantes pour le tri. L'insertion d'une nouvelle valeur et son tri sont réalisés en $O(log_2 n)$, n étant le nombre d'éléments dans l'arbre. De plus, changer la valeur d'un nœud entraîne un tri qui est également réalisé en $O(log_2 n)$. Ces deux caractéristiques sont particulièrement intéressantes pour le stockage des valeurs des arcs.

En effet, si l'insertion des valeurs dans le tas binaire est effectuée une seule fois à l'initialisation de la matrice, la descente de sous-gradient modifie régulièrement les valeurs des arcs lors de la mise à jour des multiplicateurs lagrangiens. Il faut donc que le tri soit très efficace, ce que permet un tas binaire.

Dans cette version de la procédure $GetArcOut(i,k)$, un tas binaire est associé à chaque triplet (i,k,l), c'est-à-dire à un sommet (i,k) en interaction avec la colonne l. Ainsi, le choix des arcs sortants du sommet (i,k) s'effectue en un temps linéaire puisqu'il suffit de parcourir les tas binaires associés au sommet (i,k) en prenant à chaque fois l'arc à la racine du tas. Le calcul des arcs sortants par cette méthode est de complexité linéaire $O(m_l)$ alors que la version programmation dynamique est de complexité $O(m_l n_l)$.

Cette méthode de calcul des arcs sortants est notée BH (*Binary Heap*) dans la suite du document.

5.4 Paramètres de la descente de sous-gradient

Dans la descente de sous-gradient permettant de résoudre le problème dual du lagrangien, le *pas* utilisé à chaque itération est calculé comme suit :

$$\lambda^{k+1} = \lambda^k + \frac{s^k \cdot \epsilon_k(\bar{\eta} - \eta^k)}{||s^k||^2}$$

Dans cette formule, s^k est le sous-gradient à l'itération k, η^k est la valeur du problème relaché à l'iteration k, donc la borne inférieure. Les sommets activés par la solution du problème relaché sont utilisés pour générer une solution faisable, c'est-à-dire respectant les contraintes de EM1. Le score de cette solution est utilisé comme borne supérieure. $\bar{\eta}$ est la meilleure borne supérieure trouvée dans les k itérations précédentes, c'est-à-dire la solution faisable ayant le score le plus bas.

Le paramètre ϵ_k est fixé à $0,5$ au début de l'algorithme. Si aucune amélioration n'est observée au bout de 10 itérations, ϵ est réduit d'un dixième. Par défaut, le nombre d'itérations est fixé à 3000 et le temps de calcul est limité à 600 secondes.

5.5 Résultats

Afin de mesurer les performances des différents algorithmes proposés ci-dessus et de les comparer aux résultats obtenus en utilisant CPLEX, nous avons utilisé le jeu de tests présenté en section 4.3.1. Nous mesurons les performances en termes en temps de calcul et de distance relative à l'optimale (voir section 4.3.2). Ces mesures sont tout d'abord réalisées à la racine de l'arbre de recherche.

5.5.1 Résultats à la racine de l'arbre de recherche

5.5.1.1 Temps de calcul

Nous avons tout d'abord comparé les résultats obtenus à la racine de l'arbre de recherche. Pour les calculs effectués avec CPLEX, nous considérons les temps de calculs du premier noeud de l'arbre, cela correspond à une relaxation linaire du problème. Cette méthode est notée $root_{LP}$. Pour les calculs effectués avec notre algorithme, nous arrêtons également les calculs au premier noeud de l'arbre, ce qui correspond à résoudre la relaxation lagrangienne du problème. Comme nous avons deux algorithmes permettant de calculer les arcs sortants d'un sommet, nous notons $root_{BH}$ le calcul avec l'algorithme BH (tas binaire), et $root_{DP}$ le calcul avec l'algorithme DP (programmation dynamique).

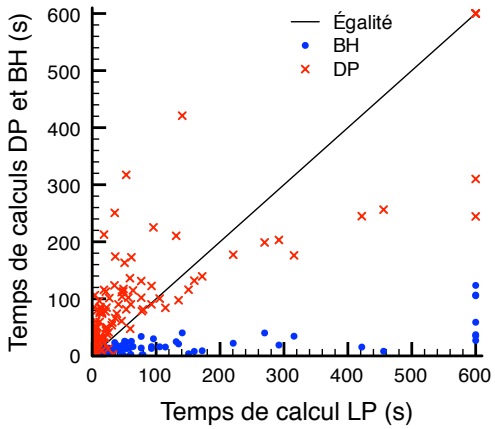

FIG. 5.12 – **Comparaison des temps de calcul au premier nœud**. Un point représente une instance (un alignement). Chaque instance est placée sur le graphique en fonction du temps de calcul obtenu par la méthode $root_{LP}$ en abscisse, et en fonction du temps de calcul des méthodes $root_{BH}$ (points marrons) et $root_{DP}$ (croix rouges) en ordonnée. Pour chaque méthode, le calcul est effectué à la racine de l'arbre de recherche. Si un point est en-dessous de la ligne d'égalité, alors la méthode $root_{DP}$ ou $root_{BH}$ est plus rapide que CPLEX. Nous remarquons que la méthode $root_{BH}$ est plus rapide que les deux autres.

Les résultats, présentés dans la figure 5.12, montrent que l'algorithme utilisant les tas binaires $root_{BH}$ semble le plus rapide. Cela est cohérent avec la définition des algorithmes puisque la complexité de BH est inférieure à celle de DP. Pour certaines instances, les temps de calculs de $root_{DP}$ sont plus longs que ceux de CPLEX. Afin de se rendre compte de la différence entre $root_{BH}$ et $root_{DP}$, nous avons comparé les distributions de leurs temps de calcul (figure 5.13).

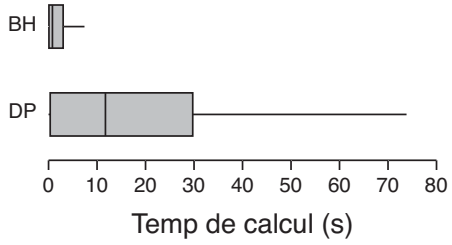

FIG. 5.13 – **Comparaison des distributions des temps de calculs de $root_{BH}$ et $root_{DP}$**. Les temps de calculs de $root_{BH}$ sont nettement inférieurs à ceux de $root_{DP}$.

Nous remarquons que $root_{BH}$ est bien plus rapide que $root_{DP}$. En particulier, le temps de calcul maximum de $root_{BH}$ est de 123,57 secondes alors que $root_{DP}$ atteint la limite fixée à 600 secondes.

5.5.1.2 Distance relative à l'optimal

Nous avons ensuite comparé les distances relatives à l'optimal des méthodes $root_{LP}$, $root_{BH}$ et $root_{DP}$. Les résultats, présentés dans la figure 5.14, montrent que la méthode $root_{DP}$ donne les distances relatives les plus fines. Cela est cohérent avec la définition des algorithmes puisque DP est plus contraint que BH.

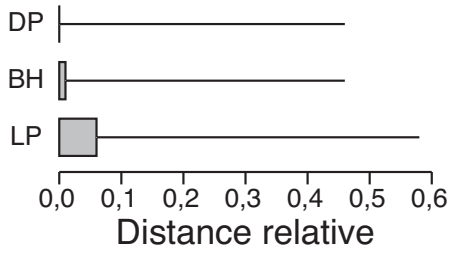

FIG. 5.14 – **Distributions des distances relatives au premier nœud de l'arbre de recherche pour les méthodes $root_{LP}$, $root_{BH}$ et $root_{DP}$.** Nous observons que la méthode la plus fine est $root_{DP}$.

Nous avons également mesuré le nombre de fois où la solution optimale est trouvée à la racine de l'arbre (tableau 5.1. Les résultats montrent que les méthodes $root_{BH}$ et $root_{DP}$ ont des taux très proches et supérieurs à celui obtenu avec CPLEX.

	$root_{LP}$	$root_{BH}$	$root_{DP}$
Taux de solution optimale	56%	72%	74%

TAB. 5.1 – **Taux de solutions optimales trouvées au premier nœud de l'arbre de recherche**.

5.5.2 Résultats dans l'arbre de recherche

Après avoir comparé les résultats obtenus à la racine de l'arbre de recherche, nous avons souhaité observer le comportement de l'algorithme dans l'arbre de recherche. Pour cela, nous avons laissé l'algorithme parcourir l'arbre, en limitant les temps de calculs à 600 secondes, sur le jeu de test présenté en section **??**.

Nous avons tout d'abord observé les temps de calcul de notre algorithme, avec la méthode BH et la méthode DP, que nous avons comparé aux temps de calcul du modèle EM1 résolu par CPLEX. Les résultats, présentés dans la figure 5.15, montrent que les temps de calcul de notre algorithme sont plus longs que ceux de CPLEX, quelque soit la méthode utilisée (BH ou DP). Ceci montre que notre algorithme converge moins

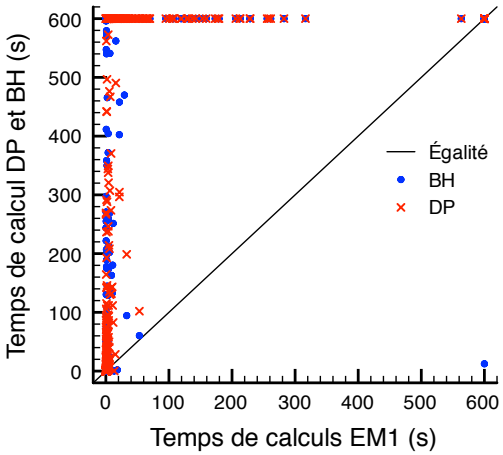

FIG. 5.15 – **Comparaison des temps de calculs de EM1, BH et DP**. La limite en temps de calcul est fixée à 600 secondes. Un point représente une instance. Les points sont placés sur le graphique en fonction du temps de calcul obtenu sur le modèle EM1 en abscisses, et en fonction du temps de calcul de la méthode BH (en bleu) et DP (en rouge) en ordonnées.

rapidement vers la solution optimale. Afin d'améliorer la convergence, des réglages seront nécessaires dans la descente de sous-gradient et dans la méthode par séparation–évaluation.

Nous nous sommes alors intéressé au pourcentage d'instances pour lesquelles la solution optimale est obtenue en moins de 600 secondes et en fonction du nombre de nœuds parcourus dans l'arbre de recherche. Le tableau 5.2 présente les résultats obtenus par les méthodes EM1, BH et DP. Nous observons de nouveau ce problème de convergence lorsque notre algorithme parcourt l'arbre de recherche. Au niveau de la distance relative à l'optimal, nos deux méthodes (DP et BH) donnent des résultats équivalents comme présenté dans la figure 5.16.

	nombre de nœuds			pas de limite
	1	10	100	de nœuds
EM1	56%	83%	87%	93%
BH	72%	78%	81%	87%
DP	74%	80%	81%	87%

TAB. 5.2 – **Taux de solutions optimales trouvées après 1, 10 et 100 nœuds de l'arbre de recherche ou au bout de 600 secondes**. Pour le modèle EM1 résolu par CPLEX, nous avons seulement les valeurs pour le nœud racine et au bout de 600 secondes. Nous observons que les méthodes BH et DP présentent de meilleurs taux que EM1 au premier nœud. Par contre, si il n'y a pas de limite dans le nombre de nœuds, CPLEX a un meilleur taux.

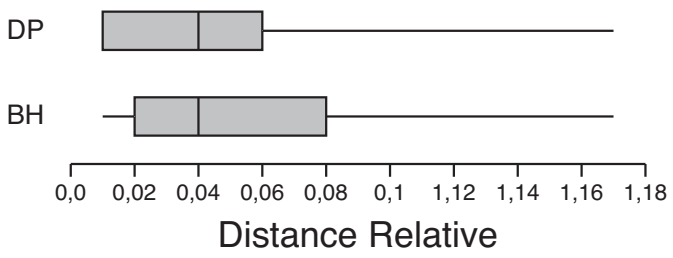

FIG. 5.16 – **Distribution des distances relatives des méthodes BH et DP après parcours dans l'arbre de recherche**. La limite en temps de calcul est fixée à 600 secondes. Nous remarquons que les distributions sont très proches, il n'y a pas de différences notables entre les deux méthodes. Pour des raisons de lisibilités, nous avons retiré du jeu de test toutes les instance où la distance relative est nulle.

5.6 Ce qu'il faut retenir de ce chapitre

Afin de résoudre rapidement des alignements locaux pour la reconnaissance de repliements, nous proposons d'utiliser une méthode par séparation–évaluation dans laquelle les bornes des sous-problèmes sont générées par une relaxation lagrangienne du modèle EM1 présenté dans la section précédente.

La méthode par séparation–évaluation découpe le problème en choisissant une colonne. Elle génère ainsi deux sous-problèmes, dans le premier, la colonne choisie est active, dans le second, la colonne choisie est inactive. Lorsque toutes les colonnes sont choisies, les problèmes sont découpés à partir d'un sommet pivot.

La relaxation lagrangienne du modèle EM1, notée ER1, consiste à relacher les contraintes (4.6) et (4.8) qui sont intégrées à la fonction objectif du modèle. Le problème dual de la relaxation lagrangienne est ensuite résolu par une descente de sous-gradient.

Afin de résoudre le problème relaché ER1, nous avons implémenté un algorithme par programmation dynamique dans lequel les mouvements diagonaux sont contraints par la taille des blocs de la structure. Pour chaque case de la programmation dynamique, il est nécessaire de trouver les meilleurs arcs sortants du sommet correspondant. Nous avons implémenté deux méthodes permettant de trouver les meilleurs arcs sortants. La première, notée DP, est basée sur une programmation dynamique locale et elle garantit que les contraintes d'ordre et de non-chevauchement sont respectées par les extrémités droites des arcs. La deuxième méthode, notée BH, ne garantit pas le respect des contraintes précédentes mais est plus rapide car elle utilise des tas binaires.

Sur le premier nœud de l'arbre de recherche, les expérimentations montrent que notre algorithme utilisant la méthode BH est plus rapide que CLPEX pour résoudre le modèle EM1. De plus, notre algorithme trouve la solution optimal du problème pour 72% des

instances alors que CPLEX n'en trouve que 56%. Par contre, si nous laissons l'algorithme parcourir l'arbre de recherche, les temps de calculs explosent. Notre algorithme ne converge pas rapidement vers la solution optimale du problème et des réglages seront nécessaires afin de corriger ce défaut.

Chapitre 6

Observations sur les alignements locaux et perspectives

6.1 Comparaison avec les alignements globaux

Dans le chapitre 4, nous avons présenté deux protéines (1ukyA et 1dvrA) dont les structures sont similaires à 77% (avec l'outil TopMatch[1]). Néanmoins, 1dvrA contient trois feuillets-β de plus que 1ukyA. Ces feuillets-β surnuméraires entrainent un décalage lors de l'alignement global entre la séquence de 1ukyA et la structure de 1dvrA. Cet alignement est illustré dans la figure 4.3 que nous reproduisons ici.

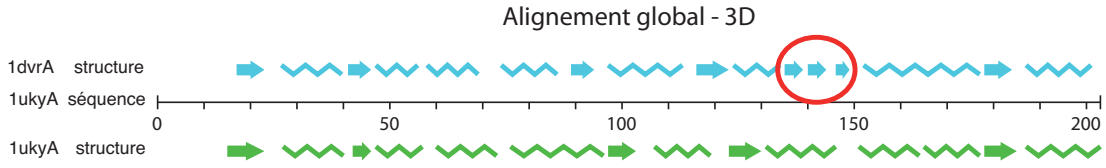

FIG. 6.1 – **Alignement global entre la séquence de 1ukyA et la structure de 1dvrA**. La séquence de 1ukyA est représentée par un trait noir, la structure de 1dvrA est en bleu. Les hélices-α sont représentées par des zigzags, les feuillets-β par des flèches. La structure de 1ukyA est représentée en vert sous la séquence. Les feuillets-β entourés en rouge provoquent un décalage des éléments de structure secondaire précédents.

Dans un alignement local, nous espérons que ces feuillets-β surnuméraires vont être omis, permettant ainsi un alignement optimal des autres SSEs de 1dvrA sur la séquence de 1ukyA. Nous avons tout d'abord testé un alignement local utilisant la fonction 3D définie en section 3.3.2. L'alignement local obtenu est présenté dans la figure 6.2.

[1] http ://topmatch.services.came.sbg.ac.at/

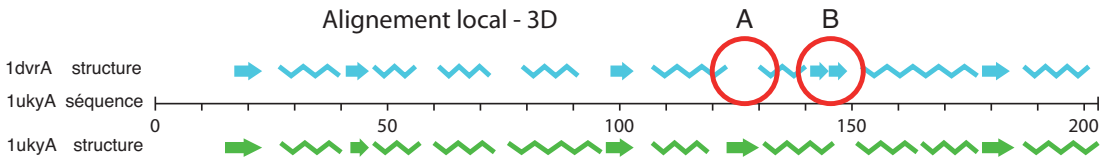

FIG. 6.2 – **Alignement local entre la séquence de 1ukyA et la structure de 1dvrA**. La séquence de 1ukyA est représentée par un trait noir, la structure de 1dvrA est en bleu. Les hélices-α sont représentées par des zigzags, les feuillets-β par des flèches. La structure de 1ukyA est représentée en vert sous la séquence. La fonction de score utilisée est la fonction 3D. **A**) Un feuillet-β a été enlevé dans 1dvrA alors qu'il y en a un dans 1ukyA à cette position. **B**) Un feuillet a été enlevé. Cela a réduit le décalage des blocs précédents.

Nous observons tout d'abord qu'un des feuillets surnuméraires a été enlevé (figure 6.2.B). Cela a permis aux blocs précédents d'être alignés sans décalage, en face des SSEs correspondantes. Néanmoins, un bloc a été enlevé alors que cela ne semble pas nécessaire (figure 6.2.A). Nous pouvons supposer que cette délétion est due à la fonction de score 3D. Pour vérifier cette hypothèse, nous avons de nouveau aligné 1ukyA et 1dvrA avec un alignement local mais en utilisant la fonction de score SSE. L'alignement obtenu est présenté dans la figure 6.3.

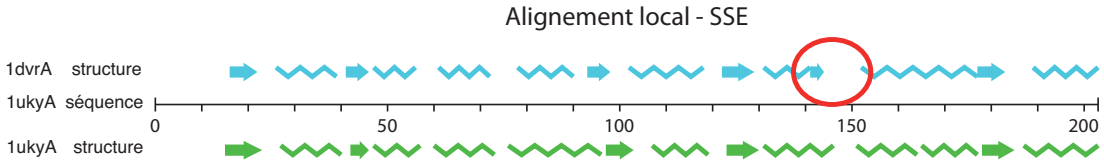

FIG. 6.3 – **Alignement local, utilisant la fonction SSE, entre la séquence de 1ukyA et la structure de 1dvrA**. La séquence de 1ukyA est représentée par un trait noir. Dans le cercle rouge, les feuillets-β qui provoquaient un décalage ont été omis.

Dans cet alignement, deux des trois feuillets ont été omis, permettant ainsi un meilleur alignement des blocs le long de la séquence. La délétion de la figure 6.2.A est donc bien due à la fonction 3D puisque la fonction SSE ne l'omet pas. Notre algorithme d'alignement local permet donc bien d'omettre des blocs quand la fonction de score le nécessite.

Néanmoins, l'exemple de la figure 6.2 illustre bien l'influence de la fonction de score sur la délétion des blocs et pose plusieurs problèmes qui sont abordés dans les sections suivantes.

6.2 Délétions : quelle limite ?

Dans un alignement local, nous avons supposé qu'un bloc est omis si sa contribution au score est inférieur à 0. Or, nous ne savons pas si cette limite est justifiée. Pour cela, il faudrait connaître les scores de chaque bloc à chaque position puis vérifier que les blocs "mal positionnés" ont un score inférieur 0. Encore faut-il savoir ce que signifie "mal positionnés". Et comme le score d'un bloc dépend aussi des positions des autres blocs, la combinatoire engendrée complexifie cette évaluation. Il n'est donc pas certain que 0 soit la bonne limite et il est difficile de déterminer cette limite.

De plus, pour qu'une telle limite existe, cela suppose que le score et la similarité soient proportionnels. Or, ce n'est pas le cas pour les fonctions de FROSTO. Notre algorithme étant programmé pour omettre des blocs si leur contribution est négative, des développements sont nécessaires afin d'adapter les fonctions de score et de les rendre plus discriminantes.

6.3 Définition des blocs

La définition des blocs est également un élément critique dans le comportement des alignements locaux. En effet, les blocs sont censés représenter les zones conservées dans la structure des protéines. Dans un premier temps, nous avons choisi de définir les blocs sur les éléments de structure secondaire. Or dans certaines familles de protéines, nous savons que certains SSEs ne sont pas conservés dans toutes les structures. De plus, les SSEs n'ont pas toujours la même longueur d'une protéine à l'autre.

Ainsi, pour définir les blocs d'une structure sur des SSEs, il faudrait vérifier la longueur et la présence de ces SSEs dans la famille de cette structure ou générer un consensus sur un alignement multiple de structures.

Un autre possibilité est de revenir à la définition d'un bloc. Un bloc est une zone conservée au cours de l'évolution. La zone peut-être conservée au niveau de la séquence ou de la structure. Nous pourrions donc combiner des informations de séquence avec des informations de structure. Nous pourrions également réduire la longueur des blocs à 1, c'est-à-dire utiliser chaque acide aminé comme bloc. Néanmoins, dans ce cas, la combinatoire explose et le problème devient très difficile à résoudre.

La définition des blocs est un sujet sur lequel nous avons collaboré avec François Coste de l'équipe Symbiose et Joël Pothier de l'université Paris 6. C'est un travail en cours au moment de la rédaction de cette thèse.

6.4 Fonction de score

Dans le premier exemple présenté dans ce chapitre, nous avons remarqué qu'un bloc disparaissait alors qu'il paraissait raisonnable de l'aligner. Nous avons vu que ce comportement était dû à la fonction de score 3D. Cette fonction de score mesure la quantité d'information dans les colonnes d'un alignement multiple. Cette définition se heurte à plusieurs écueils.

Le premier écueil est le nombre de séquences dans l'alignement multiple. En effet, dériver des scores à partir 4 ou de 300 séquences n'a pas la même signification. Nous devons donc recruter un certain nombre de séquences lors de l'alignement multiple pour avoir des statistiques fiables.

Un deuxième écueil est d'avoir de la redondance dans l'alignement multiple. En effet, si la moitié des séquences de l'alignement multiple partagent plus de 60% d'identité de séquence, nous introduisons un biais dans les statistiques. Il est donc nécessaire de filtrer les alignements mutilples afin de réduire la redondance. Néanmoins, cela réduit le nombre de séquences dans les alignements multiples et nous retrouvons le premier écueil.

Afin de peupler les alignements multiples sans ajouter de redondance, nous pouvons les enrichir avec des alignements mutliples de structures. Comme la structure est plus conservée que la séquence au cours de l'évolution, nous allons ainsi ajouter des séquences qui partagent moins de 30% d'identité de séquence mais qui se replient dans la même structure.

Afin de remplacer les fonctions de scores basées sur les statistiques, nous pouvons aussi utiliser des fonctions de scores basées sur les propriétés physiques, comme les champs moyens, ou conformationnelles, comme l'accessibilité au solvant. Nous pouvons également ajouter des contraintes plus simples comme interdire à des acides aminés hydrophobes de s'aligner à des positions hydrophiles. Ce travail exploratoire sur l'enrichissement des scores 3D et le développement de nouvelles fonctions de score pour FROSTO est en cours au moment de la rédaction de ce document.

6.5 Comparaison entre alignements locaux

Dans FROST, les alignements globaux sont comparés grâce à un score normalisé. Pour un alignement entre une séquence a et une structure b, le score normalisé est évalué grâce au calcul d'une distribution empirique des "mauvais" scores, c'est-à-dire des scores correspondant aux séquences qui ne se replient pas dans la structure b. Pour cela, 200 séquences aléatoires, de même longueur que la séquence a, sont générées. Ces 200 séquences sont alignées avec la structure b. Nous obtenons ainsi 200 scores pour

des séquences qui ne se replient pas dans la structure b. Ce score normalisé permet de comparer des alignements globaux, ce qui n'est pas possible avec le score brut.

Dans le cas des alignements locaux, si nous utilisons la même procédure, nous n'obtenons pas les mêmes résultats. En effet, une structure est définie par ses blocs et les interactions entre ses blocs. Si un bloc est omis lors de l'alignement, alors ce n'est plus vraiment la même structure. Ainsi, si nous alignons une structure contre 200 séquences avec un algorithme d'alignement local, nous obtenons en fait 200 alignements dans lesquels les structures sont différentes. Nous ne mesurons plus les "mauvais" scores pour 200 séquences contre une structure, mais pour 200 séquences contre n structures, n pouvant varier grandement. La statistique permettant de calculer le score normalisé est alors faussée.

Afin d'adapter la normalisation aux alignements locaux, nous pourrions utiliser la structure b' obtenue dans un alignement. Si des blocs sont omis lors de l'alignement, nous les enlevons de la structure b afin d'obtenir la structure b'. Cette structure b' est ensuite alignée avec 200 séquences aléatoires par un algorithme d'alignement global. Ainsi, nous mesurons bien les "mauvais" scores de la structure b'. Cette procédure n'est pas encore implémentée dans FROSTO à la rédaction de ce document.

6.6 Ce qu'il faut retenir de ce chapitre

Les alignements locaux pour la reconnaissance de repliements permettent les délétions dans la structure alignée. Ces délétions permettent de détecter des similarités locales qui ne pouvaient être détectées par un algorithme d'alignement global. Néanmoins, les délétions dépendent complètement de la fonction de score et de la définition des blocs utilisées. Ainsi, la fonction de score 3D omet des blocs à des positions où la fonction de score SSE les aligne. Il est donc nécessaire de développer des fonctions de score adaptées aux alignements locaux et de trouver une définition alternative des blocs utilisés dans FROSTO.

Conclusion

Dans ce travail de thèse, notre objectif était de proposer une méthode d'alignement local utilisant des informations pairées dans lequel les délétions sont possibles. À notre connaissance, un outil proposant ce type d'alignement n'était pas disponible.

Basé sur les travaux d'Andonov et al. [4] et du logiciel FROST développé par Marin et al. [44], nous avons développé le logiciel FROSTO qui permet de réaliser des alignements locaux utilisant des informations pairées. Ces alignements étant indépendants les uns des autres, nous avons parallélisé leurs exécutions comme présenté dans le chapitre 3

Pour réaliser ces alignements locaux, nous avons formalisé le concept d'alignement local à l'aide d'un graphe d'alignement local présenté dans le chapitre 4. Puis, nous avons développé 5 modélisations des alignements locaux en programmation linéaire en nombres entiers. Ces modèles ont été implémentés dans FROSTO à l'aide de la librairie de fonctions du logiciel CPLEX 10.0. Le premier modèle est appelé modèle compact (CM). Les modèles suivants sont appelés modèles étendus (EM1, EM2, EM3 et EM4).

Nous avons comparé ces modèles en terme de temps de calcul et de distance relative à l'optimal. Les expérimentations montrent que EM3 est le modèle le plus fin mais aussi le plus long à résoudre. Par contre, EM1 est le modèle le plus rapide et présente une distance relative à l'optimal raisonnablement fine.

Afin de résoudre ces modèles efficacement, nous avons ensuite développé un algorithme dédié basé sur une relaxation lagrangienne du modèle EM1 intégrée dans un arbre de recherche par séparation–évaluation. Cet algorithme est présenté dans le chapitre 5. Le problème relaché est résolu par une programmation dynamique dans laquelle la valeur d'une case peut être calculée de deux façons. La première est d'utiliser une programmation dynamique locale (DP) et la deuxième est d'utiliser des tas binaires (BH). La méthode DP est de complexité quadratique alors que la méthode BH est de complexité linéaire. Les expériences montrent qu'au premier nœud de l'arbre de recherche, l'utilisation de BH permet de résoudre le problème relaché plus rapidement qu'avec CPLEX ou avec DP. De plus, la distance relative à l'optimal de notre algorithme, quelque soit l'option de calcul, est plus fine que celle trouvé par CLPEX. En particulier, dans 74% des cas, la solution optimale est trouvée au premier nœud par la méthode DP (72% pour BH).

Par contre, si nous laissons l'algorithme parcourir l'arbre de recherche, nous ne trouvons pas la solution optimale en un temps raisonnable. Des réglages seront donc nécessaires afin que notre algorithme converge rapidement vers la solution optimale.

Enfin, nous avons observé le comportement des alignements locaux sur des protéines. Les expériences montrent que des délétions ont bien lieu lors d'un alignement local utilisant des informations pairées. Néanmoins, si nous utilisons la fonction de score 3D, des délétions apparaissent à des positions où les structures secondaires sont similaires. Ce comportement illustre plusieurs sources d'améliorations possibles, en particulier au niveau de la définition des blocs structuraux et des fonctions de scores.

FROST était un logiciel permettant de réaliser des alignements globaux en utilisant deux fonctions de scores (1D et 3D). FROSTO est un outil qui permet désormais tous les types d'alignements (globaux, semi-globaux et locaux) sur trois fonctions de score (1D, 3D et SSE) utilisant des informations locales ou pairées. Nous pouvons donc aligner une grande structure sur une petite séquence afin de mesurer des similarités locales. Grâce à cet outil, nous détectons des homologies lointaines qui ne sont pas détectables par des méthodes d'alignements séquence–séquence. Tous les types d'alignements étant disponibles, la qualité de la détection, c'est-à-dire la sensibilité et la spécificité de FROSTO, sera conditionnée par la qualité des fonctions de scores et une représentation des structures de protéines plus significative. Notre outil pourra alors s'intégrer comme première étape dans un processus de prédiction de la structure des protéines.

Bibliographie

[1] Tatsuya Akutsu and Satoru Miyano. On the approximation of protein threading. *Theoretical Computer Science*, 1998.

[2] S F Altschul, W Gish, W Miller, E W Myers, and D J Lipman. Basic local alignment search tool. *J Mol Biol*, 215(3) :403–10, 1990.

[3] S F Altschul, T L Madden, A A Schäffer, J Zhang, Z Zhang, W Miller, and D J Lipman. Gapped blast and psi-blast : a new generation of protein database search programs. *Nucleic Acids Res*, 25(17) :3389–402, 1997.

[4] Rumen Andonov, Stefan Balev, and Nicola Yanev. Protein Threading : From Mathematical Models to Parallel Implementations. *INFORMS JOURNAL ON COMPUTING*, 16(4) :393–405, 2004.

[5] C B Anfinsen. Principles that govern the folding of protein chains. *Science*, 181(96) :223–30, 1973.

[6] S Balev. Solving the protein threading problem by lagrangian relaxation. *Lecture Notes in Computer Science*, 2004.

[7] Moshe Ben-David, Orly Noivirt-Brik, Aviv Paz, Jaime Prilusky, Joel L Sussman, and Yaakov Levy. Assessment of casp8 structure predictions for template free targets. *Proteins*, 77 Suppl 9 :50–65, 2009.

[8] Dennis A Benson, Ilene Karsch-Mizrachi, David J Lipman, James Ostell, and Eric W Sayers. Genbank. *Nucleic Acids Res*, 38 :D46–51, 2010.

[9] J U Bowie, R Lüthy, and D Eisenberg. A method to identify protein sequences that fold into a known three-dimensional structure. *Science*, 253(5016) :164–70, 1991.

[10] S E Brenner, C Chothia, and T J Hubbard. Assessing sequence comparison methods with reliable structurally identified distant evolutionary relationships. *Proc Natl Acad Sci USA*, 95(11) :6073–8, 1998.

[11] N-V Buchete, J E Straub, and D Thirumalai. Development of novel statistical potentials for protein fold recognition. *Curr Opin Struct Biol*, 14(2) :225–32, 2004.

[12] P. M Camerini, L Fratta, and F Maffioli. On improving relaxation methods by modified gradient techniques. *Math. Programming Stud.*, (3) :26–34, 1975.

[13] Vijayalakshmi Chelliah, Tom Blundell, and Kenji Mizuguchi. Functional restraints on the patterns of amino acid substitutions : application to sequence-structure homology recognition. *Proteins*, 61(4) :722–31, 2005.

[14] E. W Cheney and A. A Goldstein. Newton's method for convex programming and tchebycheff approximation. *Numer. Math.*, 1 :253–268, 1959.

[15] C Chothia. Proteins. one thousand families for the molecular biologist. *Nature*, 357(6379) :543–4, 1992.

[16] C Chothia and A M Lesk. The relation between the divergence of sequence and structure in proteins. *EMBO J*, 5(4) :823–6, 1986.

[17] Melissa S Cline, Kevin Karplus, Richard H Lathrop, Temple F Smith, Robert G Rogers, and David Haussler. Information-theoretic dissection of pairwise contact potentials. *Proteins*, 49(1) :7–14, 2002.

[18] Andrew F W Coulson and John Moult. A unifold, mesofold, and superfold model of protein fold use. *Proteins*, 46(1) :61–71, 2002.

[19] Gavin E Crooks and Steven E Brenner. An alternative model of amino acid replacement. *Bioinformatics*, 21(7) :975–80, 2005.

[20] M O Dayhoff, R M Schwartz, and B C Orcutt. A model of evolutionary change in proteins. *Atlas of protein sequence and structure*, 5(suppl 3) :345–351, 1978.

[21] Roland L Dunbrack. Sequence comparison and protein structure prediction. *Curr Opin Struct Biol*, 16(3) :374–84, 2006.

[22] Richard Durbin, S Eddy, A Krogh, and G Mitchinson. Biological sequence analysis : probabilistic models of proteins and nucleic acids. *Cambridge University Press : Cambridge*, page 356, 1998.

[23] Robert C Edgar. Muscle : multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Res*, 32(5) :1792–7, 2004.

[24] R M Fano. Transmition of information : a statistical theory of communication. *MIT Press*, 1961.

[25] J F Gibrat, T Madej, and S H Bryant. Surprising similarities in structure comparison. *Curr Opin Struct Biol*, 6(3) :377–85, 1996.

[26] Mileidy W Gonzalez and William R Pearson. Homologous over-extension : a challenge for iterative similarity searches. *Nucleic Acids Res*, 2010.

[27] Nalin C W Goonesekere and Byungkook Lee. Frequency of gaps observed in a structurally aligned protein pair database suggests a simple gap penalty function. *Nucleic Acids Res*, 32(9) :2838–43, 2004.

[28] M Guignard. Lagrangean relaxation. *TOP*, 2003.

[29] Monique Guignard and Moshe B. Rosenwein. An application-oriented guide for designing lagrangean dual ascent algorithms. *European Journal of Operational Research*, 43(2) :197 – 205, 1989.

[30] M Held and R M Karp. The Traveling-Salesman Problem and Minimum Spanning Trees. *OPERATIONS RESEARCH*, 18(6) :1138–1162, 1970.

[31] M Hendlich, P Lackner, S Weitckus, H Floeckner, R Froschauer, K Gottsbacher, G Casari, and M J Sippl. Identification of native protein folds amongst a large number of incorrect models. the calculation of low energy conformations from potentials of mean force. *J Mol Biol*, 216(1) :167–80, 1990.

[32] S Henikoff and J G Henikoff. Amino acid substitution matrices from protein blocks. *Proc Natl Acad Sci USA*, 89(22) :10915–9, 1992.

[33] Kristoffer Illergård, David H Ardell, and Arne Elofsson. Structure is three to ten times more conserved than sequence–a study of structural response in protein cores. *Proteins*, 77(3) :499–508, 2009.

[34] Lukasz Jaroszewski. Protein structure prediction based on sequence similarity. *Methods Mol Biol*, 569 :129–56, 2009.

[35] D T Jones. Protein secondary structure prediction based on position-specific scoring matrices. *J Mol Biol*, 292(2) :195–202, 1999.

[36] Jr Kelley. The cutting-plane method for solving convex programs. 8 :703–712, 1960.

[37] A Krogh, B Larsson, G von Heijne, and E L Sonnhammer. Predicting transmembrane protein topology with a hidden markov model : application to complete genomes. *J Mol Biol*, 305(3) :567–80, 2001.

[38] Remko K P Kuipers, Henk-Jan Joosten, Eugene Verwiel, Sjoerd Paans, Jasper Akerboom, John van der Oost, Nicole G H Leferink, Willem J H van Berkel, Gert Vriend, and Peter J Schaap. Correlated mutation analyses on super-family alignments reveal functionally important residues. *Proteins*, 76(3) :608–16, 2009.

[39] R H Lathrop. The protein threading problem with sequence amino acid interaction preferences is np-complete. *Protein Eng*, 7(9) :1059–68, 1994.

[40] R H Lathrop and T F Smith. Global optimum protein threading with gapped alignment and empirical pair score functions. *J Mol Biol*, 255(4) :641–65, 1996.

[41] Claude Lemarechal. An algorithm for minimizing convex functions. *Proc. IFIP'74*, pages 552–556, 1974.

[42] Xinsheng Liu, Ke Fan, and Wei Wang. The number of protein folds and their distribution over families in nature. *Proteins*, 54(3) :491–9, 2004.

[43] A Lupas. Predicting coiled-coil regions in proteins. *Curr Opin Struct Biol*, 7(3) :388–93, 1997.

[44] Antoine Marin, Joël Pothier, Karel Zimmermann, and Jean-François Gibrat. Frost : a filter-based fold recognition method. *Proteins*, 49(4) :493–509, 2002.

[45] O Du Merle, J-L Goffin, and J-P Vial. On improvements to the analytic center cutting plane method. *Comput. Optim. Appl.*, 11(1) :37–52, 1998.

[46] B Morgenstern, K Frech, A Dress, and T Werner. Dialign : finding local similarities by multiple sequence alignment. *Bioinformatics*, 14(3) :290–4, 1998.

[47] John Moult, Krzysztof Fidelis, Andriy Kryshtafovych, Burkhard Rost, and Anna Tramontano. Critical assessment of methods of protein structure prediction - round viii. *Proteins : Structure, Function, and Bioinformatics*, 77(S9) :1–4, 2009.

[48] Tobias Müller, Rainer Spang, and Martin Vingron. Estimating amino acid substitution models : a comparison of dayhoff's estimator, the resolvent approach and a maximum likelihood method. *Mol Biol Evol*, 19(1) :8–13, 2002.

[49] A G Murzin, S E Brenner, T Hubbard, and C Chothia. Scop : a structural classification of proteins database for the investigation of sequences and structures. *J Mol Biol*, 247(4) :536–40, 1995.

[50] S B Needleman and C D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol*, 48(3) :443–53, 1970.

[51] George L. Nemhauser and Laurence A. Wolsey. *Integer and combinatorial optimization*. 1999.

[52] C. A. Orengo, A. D. Michie, S. Jones, D. T. Jones, M. B. Swindells, and J. M. Thornton. Cath–a hierarchic classification of protein domain structures. *Structure (London, England : 1993)*, 5(8) :1093–1108, 1997.

[53] Jimin Pei. Multiple protein sequence alignment. *Curr Opin Struct Biol*, 18(3) :382–6, 2008.

[54] S Pietrokovski. Searching databases of conserved sequence regions by aligning protein multiple-alignments. *Nucleic Acids Res*, 24(19) :3836–45, 1996.

[55] Vincent Poirriez, Rumen Andonov, Antoine Marin, and Jean-Fran ?ois Gibrat. Frost : Revisited and distributed. *Parallel and Distributed Processing Symposium, International*, 8 :200a, 2005.

[56] Yuan Qi, Ruslan I Sadreyev, Yong Wang, Bong-Hyun Kim, and Nick V Grishin. A comprehensive system for evaluation of remote sequence similarity detection. *BMC Bioinformatics*, 8 :314, 2007.

[57] B Qian and R A Goldstein. Distribution of indel lengths. *Proteins*, 45(1) :102–4, 2001.

[58] G A Reeves, D Talavera, and J M Thornton. Genome and proteome annotation : organization, interpretation and integration. *J R Soc Interface*, 6(31) :129–47, 2009.

[59] D W Rice and D Eisenberg. A 3d-1d substitution matrix for protein fold recognition that includes predicted secondary structure of the sequence. *J Mol Biol*, 267(4) :1026–38, 1997.

[60] B Rost. Twilight zone of protein sequence alignments. *Protein Eng*, 12(2) :85–94, 1999.

[61] L Rychlewski, L Jaroszewski, W Li, and A Godzik. Comparison of sequence profiles. strategies for structural predictions using sequence information. *Protein Sci*, 9(2) :232–41, 2000.

[62] C Sander and R Schneider. Database of homology-derived protein structures and the structural meaning of sequence alignment. *Proteins*, 9(1) :56–68, 1991.

[63] J Shi, T L Blundell, and K Mizuguchi. Fugue : sequence-structure homology recognition using environment-specific substitution tables and structure-dependent gap penalties. *J Mol Biol*, 310(1) :243–57, 2001.

[64] M J Sippl. Calculation of conformational ensembles from potentials of mean force. an approach to the knowledge-based prediction of local structures in globular proteins. *J Mol Biol*, 213(4) :859–83, 1990.

[65] J Skolnick, L Jaroszewski, A Kolinski, and A Godzik. Derivation and testing of pair potentials for protein folding. when is the quasichemical approximation correct ? *Protein Sci*, 6(3) :676–88, 1997.

[66] T F Smith and M S Waterman. Identification of common molecular subsequences. *J Mol Biol*, 147(1) :195–7, 1981.

[67] Johannes Söding. Protein homology detection by hmm-hmm comparison. *Bioinformatics*, 21(7) :951–60, 2005.

[68] J D Thompson, D G Higgins, and T J Gibson. Clustal w : improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res*, 22(22) :4673–80, 1994.

[69] Guoli Wang and Roland L Dunbrack. Scoring profile-to-profile sequence alignments. *Protein Sci*, 13(6) :1612–26, Jun 2004.

[70] Z X Wang. How many fold types of protein are there in nature ? *Proteins*, 26(2) :186–91, 1996.

[71] J J Ward, J S Sodhi, L J McGuffin, B F Buxton, and D T Jones. Prediction and functional analysis of native disorder in proteins from the three kingdoms of life. *J Mol Biol*, 337(3) :635–45, 2004.

[72] P Wentges. Weighted dantzig-wolfe decomposition for linear mixed-integer programming. *International Transactions in Operational Research*, 4(2) :151–162, 1997.

[73] Markus Wistrand and Erik L L Sonnhammer. Improving profile hmm discrimination by adapting transition probabilities. *J Mol Biol*, 338(4) :847–54, 2004.

[74] Y I Wolf, N V Grishin, and E V Koonin. Estimating the number of protein folds and families from complete genome data. *J Mol Biol*, 299(4) :897–905, 2000.

[75] J C Wootton. Non-globular domains in protein sequences : automated segmentation using complexity measures. *Comput Chem*, 18(3) :269–85, 1994.

[76] Feng Xu, Pan Du, Hongbo Shen, Hairong Hu, Qi Wu, Jun Xie, and Long Yu. Correlated mutation analysis on the catalytic domains of serine/threonine protein kinases. *PLoS ONE*, 4(6) :e5913, 2009.

[77] Jinbo Xu, Ming Li, Dongsup Kim, and Ying Xu. Raptor : optimal protein threading by linear programming. *Journal of bioinformatics and computational biology*, 1(1) :95–117, 2003.

[78] Marcus A Zachariah, Gavin E Crooks, Stephen R Holbrook, and Steven E Brenner. A generalized affine gap model significantly improves protein sequence alignment accuracy. *Proteins*, 58(2) :329–38, 2005.

[79] Shijie Zhang, Wei Su, and Jiong Yang. Arcs-motif : discovering correlated motifs from unaligned biological sequences. *Bioinformatics*, 25(2) :183–9, 2009.

[80] Yang Zhang, Isaac A Hubner, Adrian K Arakaki, Eugene Shakhnovich, and Jeffrey Skolnick. On the origin and highly likely completeness of single-domain protein structures. *Proc Natl Acad Sci USA*, 103(8) :2605–10, 2006.

[81] J Zowe. Nondifferentiable optimization. *Computational mathematical programming*, 15 :323–356, 1985.

Liste des figures

Liste des tableaux

Résumé

Détecter des similarités et des homologies entre protéines est une étape cruciale du processus d'annotation des génomes. Afin de détecter des homologies, les alignements de séquences, globaux ou locaux, sont couramment utilisés. Néanmoins, dans la "zone d'ombre", nous devons utiliser les méthodes de reconnaissance de repliements pour détecter des homologies. Dans ce domaine, le problème du "Protein Threading" (PTP) utilise des paramètres pairés pour aligner globalement une séquence de protéine avec une structure de protéine. À notre connaissance, il n'existe pas de méthode d'alignement local utilisant des paramètres pairés. À partir du PTP, nous proposons cinq modélisations mathématiques de ces alignements locaux qui ont été implémentées et testées grâce au logiciel CPLEX 10.0. Nous avons ensuite développé un algorithme dédié permettant de résoudre un de ces modèles. Cet algorithme utilise des techniques connues en recherche opérationnelle : la séparation-évaluation, la descente de sous-gradient et la relaxation lagrangienne. Bien que les alignements locaux soient d'une plus grande complexité, nous montrons qu'ils sont réalisables et qu'ils améliorent la qualité des alignements.

Abstract

Detecting similarities and homologies between proteins is a key step during the annotation process. To find such homologies, multiple sequence alignments are widely used. These methods provide global to local alignment features. Nevertheless, in the so called "Twilight Zone", one must relies on fold recognition methods to find homologous proteins. In this field, the Protein Threading Problem (PTP) uses pairwise parameters to globaly align a protein sequence with a protein structure. As far as we know, no local alignment method using pairwise parameters exists. Based on the PTP, we proposed 5 mathematical formulations of such local alignments. These formulations have been implemented and tested with CPLEX 10.0 package. Then, we developed an efficient dedicated algorithm to solve local alignments. Our algorithm uses well-known techniques from Operational Research : Branch & Bound, Subgradient Descent and Lagrangian Relaxation. We show that, despite greater complexity, local alignments using pairwise parameters are feasible and improve the quality of the alignments.